Microsoft® QuickBASIC 4

Lernen und Anwenden von Microsoft QuickBASIC

Microsoft®

Microsoft® QuickBASIC

Lernen und Anwenden von Microsoft QuickBASIC

Für IBM® Personal Computer
und kompatible Computer

Microsoft Corporation

Artikelnr. 01648
Dokumentnr. 8623-400-00-3

Inhaltsverzeichnis

Teil 2: Die QuickBASIC-Programm-Entwicklungsumgebung

Teil 3: Biblotheken und Hilfsprogramme

Teil 4: Anhänge

Einleitung

Willkommen zu Microsoft® QuickBASIC, Version 4.0. QuickBASIC kombiniert eine vollständige Programmierumgebung mit der leistungsfähigsten verfügbaren BASIC-Sprache - alles, was Sie zur schnellen Entwicklung blitzschneller, kompakter Programme benötigen. In der QuickBASIC-Umgebung können Sie das Bearbeiten, Ausführen und Debuggen eines Programmes auf jede von Ihnen gewünschte Weise kombinieren, da all Ihre Programmierwerkzeuge die ganze Zeit über bereit sind und laufen. Es folgen die Gründe, warum Sie mit QuickBASIC Zeit sparen und bessere Programme schreiben:

- **Sofortige Ausführung Ihres Codes, während Sie ihn schreiben**

  QuickBASIC übersetzt jede eingegebene Zeile des Codes sofort in Anweisungen, die der Computer ausführen kann. Es gibt keinen separaten "Kompilier"-Schritt, der Sie aufhält und Ihren Gedankenfluß unterbricht. Daher können Sie Ihren Code, ungeachtet des Stadiums der Fertigstellung, jederzeit lediglich durch die Betätigung einer Taste starten. Wenn Sie mit dem Ergebnis des Programmlaufes zufrieden sind, erfordert QuickBASIC nur vier Tastenbetätigungen, um das Programm in ein selbständig ausführbares Programm zu verwandeln, das Sie aus DOS heraus starten können.

- **Sofortige Hilfe und Überprüfung der Syntax**

  Kontext-sensitive (on-line) Hilfe liefert die Syntax für jede QuickBASIC-Anweisung ebenso wie Hilfe für Menü-Befehle, Tastenkurzkombinationen sowie die "American Standard Code for Information Interchange" (ASCII)-Codes. QuickBASICs "intelligenter Editor" überprüft automatisch die Syntax jeder Zeile, während Sie diese eingeben, so daß Sie Fehler beim Schreiben Ihrer Programme sofort korrigieren können.

- **Sofortiges Debuggen**

  In QuickBASIC können Sie Ihren Code debuggen, während Sie ihn schreiben – es gibt keinen separaten Debug-Schritt. Sie müssen lediglich Ihren ausführenden Code unterbrechen, sein Verhalten analysieren, ihn bearbeiten und anschließend seine Ausführung an dem Punkt fortsetzen, an dem Sie ihn unterbrochen haben. In einem speziellen Fenster können Sie sich die wechselnden Werte von Variablen anzeigen lassen, die Sie auf drei verschiedene Arten durch das Programm verfolgen können. Mehrere Haltepunkte und Stoppbedingungen machen es leicht, neuen Code zu verändern und zu testen, weil Sie diesen direkt in Ihr laufendes Programm einfügen können.

- **Mehrmodulige Programme**

  Große Programme sind einfacher zu schreiben und zu debuggen, wenn Sie deren Teile in logisch getrennte Komponenten aufteilen. QuickBASIC macht es einfach, mehrmodulige Programme zu erstellen, weil es alle Module Ihres Programmes gleichzeitig im Speicher behält und eine sofortige Übersicht nicht nur über alle Module des Programms sondern auch über die Prozeduren, die jedes Modul enthält, bietet.

- **Mehrere Editierfenster**

  Mehrere Editierfenster in QuickBASIC erlauben es Ihnen, verschiedene Teile Ihres Programmes gleichzeitig zu betrachten. Die Verwendung mehrerer Fenster bei der Verfolgung durch Prozeduren hält die Ausführungslogik klarer, während Sie Ihren Code debuggen.

- **Volle Grafikunterstützung**

  QuickBASIC unterstützt die neuen IBM® Personal System/2™ (PS/2™)-Grafikmodi mit dem Video Graphics Array (VGA) und Multicolor Graphics Array (MCGA) genauso, wie den Farbgrafikadapter (CGA), den erweiterten Grafikadapter (EGA) sowie monochrome Modi. QuickBASIC unterstützt jetzt die Hercules® Grafikkarte.

Zusätzlich zu der Programmierumgebung, wurde mit QuickBASIC 4.0 die Sprache BASIC um folgende leistungsfähige Eigenschaften erweitert:

- QuickBASIC unterstützt Binärdateien, so daß Ihre Programme Dateien in jedem Format erstellen und manipulieren können.
- Benutzerdefinierte Datentypen vereinfachen Eingabe/Ausgabe in/aus Direktzugriffsdateien und helfen Ihnen, komplexere Datenstrukturen aus numerischen Variablen und Zeichenketten aufzubauen.
- QuickBASIC-Unterprogramme und -Funktionen können rekursiv sein (sie können sich selbst aufrufen), wodurch Sie kompakten und leistungsfähigen Code schreiben können.
- Lange (32-Bit-) Ganzzahlen sowie 8087/80287-Unterstützung bieten Ihnen schnelle und genaue Berechnungen.
- Erweiterte Anweisungen zur Ablaufsteuerung, wie zum Beispiel **SELECT CASE** und Block-**IF...THEN...ELSE**, verbessern den logischen Ablauf Ihrer Programme.

Systemanforderungen

Microsoft QuickBASIC erfordert folgende Hardware:

- Einen IBM Personal Computer oder Kompatiblen, der unter MS-DOS® oder PC-DOS, Version 2.1 oder höher, läuft
- Mindestens ein Disketten-Laufwerk (zwei werden empfohlen)
- 320 Kilobytes (K) Arbeitsspeicher

QuickBASIC-Dokumentation

Die Version 4.0 enthält eine ausführliche Dokumentation, um Ihnen ein Höchstmaß an Informationen zu bieten. Dieses Paket besteht aus drei Handbüchern:

Handbuch	*Inhalt*
Lernen und Anwenden von Microsoft QuickBASIC	Erläutert, wie Microsoft QuickBASIC auf Ihrem Computer installiert und benutzt wird. Wenn Sie noch neu auf diesem Gebiet sind, sollten Sie mit diesem Handbuch beginnen. Neben Erläuterungen zu den Grundlagen wird beschrieben, wie Sie die erweiterten Eigenschaften der QuickBASIC-Programmierumgebung einsetzen, wie zum Beispiel den "intelligenten" Editor und die Debug-Werkzeuge. Weitere Informationen finden Sie im nächsten Abschnitt, "Wie Sie dieses Handbuch verwenden".
Programmieren in BASIC: Ausgewählte Themen	Zeigt, wie Sie QuickBASIC verwenden, um übliche Programmieraufgaben durchzuführen, wie zum Beispiel Grafiken erstellen, Datendatei-E/A verwalten und komplexere Programme mit Unterprogrammen und Funktionen erstellen. Zugeschnitten auf bestimmte praktische Themen enthält dieses Handbuch viele Programmierbeispiele.
BASIC-Befehlsverzeichnis	Beschreibt jede Anweisung der Programmiersprache QuickBASIC. Erläutert werden ebenfalls die Elemente von BASIC, wie zum Beispiel Datentypen, Operatoren, Variablen, Konstanten und Ausdrücke. Die Anhänge enthalten eine ASCII-Tabelle, eine Liste der in BASIC reservierten Wörter sowie eine vollständige Liste der Fehlermeldungen, die Sie eventuell bei der Programmierung mit QuickBASIC erhalten können.

Wie Sie dieses Handbuch verwenden

Dieses Handbuch ist Ihr Hauptnachschlagewerk zur Verwendung der Programm-Entwicklungsumgebung von QuickBASIC. Es folgt eine Zusammenfassung der Teile dieses Handbuchs.

Teile	*Inhalt*
Teil 1: "Erste Schritte"	Erklärt, wie Sie QuickBASIC, Version 4.0, installieren und starten, mit besonderen Hinweisen für diejenigen, die bereits Erfahrungen mit BASICA oder früheren Versionen von QuickBASIC haben.
Teil 2: "Die Programm-Entwicklungsumgebung von QuickBASIC"	Beschreibt, wie Sie Programme innerhalb der QuickBASIC-Programmierumgebung bearbeiten, starten und debuggen. Darüber hinaus lernen Sie, wie Sie Menüs und Dialogfelder benutzen, und wie Sie Dateien verwalten.
Teil 3: "Bibliotheken und Werkzeuge"	Erläutert sowohl, wie Sie Quick-Bibliotheken erstellen, verändern und verwenden, als auch, wie Sie Programme aus DOS heraus kompilieren und binden. Es beschreibt ebenfalls, wie Sie Programme unter Verwendung des BASIC-Compilers (BC) und des Microsoft Overlay Linkers (LINK), die mit QuickBASIC geliefert werden, kompilieren und binden. Zudem wird erklärt, wie Sie Quick-Bibliotheken mit LINK erstellen, und wie Sie selbständige Bibliotheken mit dem Microsoft-Bibliotheksmanager (LIB) erstellen.
Teil 4: Anhänge	Enthalten zusätzliche hilfreiche Informationen, wie zum Beispiel eine Beschreibung der Unterschiede zwischen der QuickBASIC-Version 4.0 und älteren Versionen von QuickBASIC, Hinweise zur Konvertierung von BASICA-Programmen nach QuickBASIC und dem Aufruf von in anderen Sprachen geschriebenen Programmen aus QuickBASIC heraus sowie eine Zusammenfassung der Befehle. Ein Glossar am Ende dieses Teils definiert Begriffe, die in diesem Handbuch und in den beiden anderen Handbüchern dieser Reihe (das *BASIC-Befehlsverzeichnis* sowie *Programmieren in BASIC: Ausgewählte Themen*) verwendet werden.

Typographische Konventionen

Dieses Handbuch verwendet folgende typographische Konventionen, um Informationen über QuickBASIC zu vermitteln. Beachten Sie, daß diese Konventionen etwas einfacher sind, als diejenigen, die in den anderen Handbüchern dieser Reihe verwendet werden.

Darstellung	***Beschreibung***
`Beispiele` **`Eingabe:`**	Die in der linken Spalte gezeigte Schriftart wird zur Darstellung von Informationen benutzt, die auf Ihrem Bildschirm erscheinen. Die fettgedruckte Version dieses Schrifttyps zeigt an, daß eine Eingabe nach einem Eingabeaufforderungszeichen eingegeben wird.
Apostroph (')	Das Apostroph kennzeichnet einen Kommentar in Beispielprogrammen. Ein Apostroph wird durch Drücken der Taste eingegeben, die ein einfaches rechtes Anführungszeichen erzeugt. Die Taste kann als ' oder " beschriftet sein. Benutzen Sie nicht das einfache linke Anführungszeichen, welches durch die mit ‘ gekennzeichnete Taste erzeugt wird.
SCHLÜSSELWÖRTER	Fettgedruckte Großbuchstaben zeigen BASIC-Schlüsselwörter an. Diese Schlüsselwörter sind ein nötiger Bestandteil der Anweisungs-Syntax, es sei denn, sie sind, wie nachfolgend beschrieben, in eckige Klammern eingeschlossen. Wenn Sie Programme schreiben, müssen Sie Schlüsselwörter genau wie gezeigt eingeben. Sie können dabei Groß- und/oder Kleinbuchstaben benutzen.
Befehlsnamen	Fettgedruckte Kleinbuchstaben im Text zeigen Menü- und Befehlsnamen an.
Platzhalter	Die Begriffe in Kursivschrift sind Platzhalter für Informationen, die Sie angeben müssen, wie z. B. einen Dateinamen. Darüber hinaus wird Kursivschrift gelegentlich im Text zur Hervorhebung benutzt.
[**Wahlweise Begriffe**]	Zwischen eckigen Klammern stehende Begriffe können wahlweise verwendet werden.
{*Wahl1* | *Wahl2*}	Geschweifte Klammern und ein vertikaler Balken zeigen an, daß Sie die Wahl zwischen zwei oder mehr Begriffen haben. Sie müssen einen der Begriffe wählen, es sei denn, alle Begriffe sind auch in eckige Klammern eingeschlossen.

Darstellung	***Beschreibung***
Wiederholende Elemente...	Drei Punkte, die einem Begriff folgen, zeigen an, daß mehrere Begriffe derselben Form auftreten können.

`Programm`
.
.
.
`Fragment`
	Eine senkrechte Punktreihe zeigt an, daß ein Teil des Programms weggelassen wurde.

TASTENBEZEICHNUNGEN	Kleingedruckte Großbuchstaben werden für die Bezeichnungen von Tasten und Tastenkombinationen verwendet, wie z.B. EINGABETASTE und STRG+R. Beachten Sie bitte, daß ein Plus (+) eine Kombination von Tasten anzeigt. Zum Beispiel zeigt Ihnen STRG+E, daß die STRG-TASTE gedrückt bleiben muß, während die E-Taste betätigt wird. Die in diesem Handbuch verwendeten Tastenbezeichnungen entsprechen den Bezeichnungen, die auf den Tasten des IBM®-Personal Computers stehen. Falls Sie einen anderen Computer benutzen, können diese Tasten abweichende Bezeichnungen tragen. Die Gruppe der Cursor-Bewegungstasten (auch "Pfeiltasten" genannt), die sich im numerischen Tastenfeld rechts neben der Haupttastatur befinden, werden RICHTUNGSTASTEN genannt. Jede einzelne RICHTUNGSTASTE wird entweder durch die Richtung des Pfeiles auf der Taste bezeichnet (NACH LINKS, NACH RECHTS, NACH OBEN, NACH UNTEN) oder durch die Bezeichnung auf der Taste (BILD ↑, BILD ↓). Die Wagenrücklauftaste wird als EINGABETASTE bezeichnet.
"Definierter Ausdruck"	Anführungsstriche kennzeichnen entweder einen im Text definierten Ausdruck oder den Namen eines Begriffs in einem Dialogfeld.

Hinweis Die unten gezeigte Syntax (für die Anweisung **PRINT**) veranschaulicht viele der Darstellungskonventionen dieses Handbuches:

PRINT [*Ausdrucksliste*] [{, | ;}]

Hilfsquellen zum Lernen

Die Handbücher dieses Paketes bieten umfassende Beschreibungen zur Verwendung der QuickBASIC-Programmierumgebung. Es wird jedoch nicht der Versuch unternommen, Sie in DOS zu unterweisen, oder Sie in der Programmierung in BASIC oder Assembler zu unterrichten. Die folgenden Bücher enthalten zusätzliche Informationen zu diesen Themen. Diese Bücher werden nur zu Ihrer Information aufgeführt. Mit Ausnahme der von der Microsoft Corporation verlegten Bücher werden keine Bücher zu diesem Thema ausdrücklich empfohlen.

Bücher zu BASIC

Wenn BASIC für Sie neu ist, oder Sie zusätzliche Informationen zur Programmierung in BASIC benötigen, können die folgenden Bücher für Sie hilfreich sein:

Dwyer, Thomas A., und Margot Critchfield. *BASIC and the Personal Computer.* Reading, Mass.: Addison-Wesley Publishing Co., 1978.

Enders, Bernd, und Bob Petersen. *BASIC Primer for the IBM PC & XT.* New York, N. Y.: New American Library, 1984.

Hergert, Douglas. *Microsoft QuickBASIC.* 2. Ausgabe; Redmond, Wash.: Microsoft Press. In Vorbereitung.

Die erste Ausgabe dieses Buches erläutert Programmiertechniken für die QuickBASIC-Versionen 2.0 und 3.0. Verwenden Sie die weitergehende zweite Ausgabe für Informationen zur Programmierung mit QuickBASIC, Version 4.0.

Bücher zu Assembler

Wenn Sie Lernhilfen zur Programmierung in Assembler benötigen, können die folgenden Bücher hilfreich sein:

Lafore, Robert. *Assembly Language Primer for the IBM PC & XT.* New York, N. Y.: New American Library, 1984.

Metcalf, Christopher D., und Sugiyama, Marc B. *COMPUTE!'s Beginner's Guide to Machine Language on the IBM PC and PCjr.* Greensboro, N. C.: COMPUTE! Publications Inc., 1985.

Scanlon, Leo J. *IBM PC Assembly Language: A Guide for Programmers.* Bowie, Md.: Robert J. Brady Co., 1983.

Bücher zu DOS

Wenn Sie weitere Informationen zu DOS benötigen, lesen Sie folgende Bücher:

Duncan, Ray. *Advanced MS-DOS*. Redmond, Wash.: Microsoft Press, 1986.

Wolverton, Van. *Running MS-DOS*. 2. Ausgabe; Redmond, Wash.: Microsoft Press, 1985.

Wolverton, Van. *Supercharging MS-DOS*. Redmond, Wash.: Microsoft Press, 1986.

Vorschläge, Anregungen

Wenn Sie Anregungen oder Vorschläge bezüglich eines der zu diesem Produkt gehörenden Handbücher haben, verwenden Sie bitte die Karte zur Dokumentations-beurteilungs am Ende dieses Handbuches.



Teil 1: Erste Schritte

Willkommen zu Microsoft QuickBASIC Version 4.0. QuickBASIC bietet eine komfortable, integrierte Programmierumgebung mit all den Werkzeugen, die Sie zur schnellen Entwicklung kompakter, schneller Programme benötigen. Dieser Teil des Handbuches beschreibt, wie Sie QuickBASIC auf Ihrem Computer installieren und mit dem Einsatz von QuickBASIC beginnen, mit besonderen Hinweisen für BASICA-Programmierer und für diejenigen, die bereits mit BASIC-Compilern vertraut sind.



1 QuickBASIC installieren

Dieses Kapitel beschreibt, wie Sie QuickBASIC installieren. Wenn Sie dieses Kapitel gelesen haben, werden Sie wissen, wie Sie folgendes durchführen können:

- Kopieren (oder "Sichern") der Originaldisketten.
- Installieren der QuickBASIC-Software auf Ihrem Festplatten- oder Diskettensystem.
- Setzen der Umgebungsvariablen, so daß QuickBASIC alle Dateien, die es benötigt, finden kann.

Wenn Sie mit einer der in diesem Kapitel erwähnten DOS-Prozeduren nicht vertraut sind, schlagen Sie in Ihrer DOS-Dokumentation nach.

1.1 Sichern Ihrer Disketten

Der erste Schritt, den Sie nach dem Auspacken Ihres Paketes durchführen sollten, ist das Anlegen von Arbeitskopien Ihrer QuickBASIC-Originaldisketten, indem Sie den DOS-Befehl DISKCOPY verwenden. Legen Sie die Originaldisketten an einem sicheren Ort ab, und verwenden Sie von nun an die Kopien. Sollten die Kopien jemals beschädigt oder zerstört werden, können Sie neue Kopien von den Originaldisketten erstellen.

1.2 Disketteninhalt

Die folgende Tabelle beschreibt die Dateien, die mit diesem Produkt geliefert werden. Ihre Originaldisketten enthalten zusätzliche Dateien, u.a. BASIC-Programme (Dateien, die die Erweiterung .BAS haben) und andere Dateien, die nach dem Druck dieses Handbuches hinzugefügt wurden. Eine vollständige Liste aller Dateien sowie eine Beschreibung, auf welcher Originaldiskette sich die Dateien jeweils befinden, finden Sie auf der Originaldiskette 1 in der Datei PAKET.LST.

Dateiname	*Beschreibung*	*Unterstützte Aufgaben*
QB.EXE	Die Microsoft QuickBASIC-Programm-Entwicklungs-umgebung	Erstellen, Debuggen und Ausführen von BASIC-Programmen im Speicher; Erstellen von selbständig lauffähigen Programmen; Erstellen von Quick-Bibliotheken
BC.EXE	Der BASIC-Befehlszeilen-Compiler, der durch den Befehl **EXE-Datei erstellen** aus dem Menü **Ausführen** aufgerufen wird	Erstellen ausführbarer Programme aus QuickBASIC und DOS heraus; Erstellen von Quick-Bibliotheken
SETUP.BAT	QuickBASIC-Installations-programm	QuickBASIC installieren
QB.QLB	Quick-Bibliothek, die Unterstützungsroutinen für DOS-Interrupts enthält	
QB.LIB	Selbständige Bibliothek, die Unterstützungsroutinen für DOS-Interrupts enthält	Erstellen von Quick-Bibliotheken
QB.BI	Include-Datei zur Verwendung mit QB.QLB und QB.LIB	Siehe Kapitel 8, "Quick-Bibliotheken"
INFO.DOC	Wichtige Informationen, die erst nach dem Druck dieses Handbuches verfügbar wurden	
PAKET.LST	Liste aller Dateien, die sich auf den Originaldisketten befinden	
BRUN40.EXE	QuickBASIC-Laufzeitmodul	Ausführung ausführbarer BASIC-Programme, die mit BRUN40.LIB erstellt wurden
BQLB40.LIB	Laufzeitunterstützung für Quick-Bibliotheken	Erstellen von Quick-Bibliotheken
BRUN40.LIB	QuickBASIC-Laufzeitmodul-Bibliothek	Erstellen ausführbarer Programme aus QuickBASIC und DOS heraus

Dateiname	*Beschreibung*	*Unterstützte Aufgaben*
BCOM40.LIB	Alternative QuickBASIC-Laufzeitbibliothek	Erstellen ausführbarer Programme aus QuickBASIC und DOS heraus. Ausführbare Programme, die mit dieser Bibliothek erstellt wurden, erfordern zur Ausführung nicht BRUN40.EXE.
LINK.EXE	Der Microsoft Overlay Linker	Erstellen ausführbarer Programme aus QuickBASIC und DOS heraus; Erstellen von Quick-Bibliotheken
LIB.EXE	Der Microsoft-Bibliotheks-Manager	Erstellen selbständiger Programme
NOCOM.OBJ	Objektdatei zum Binden mit Programmen, die keine Unterstützung für Datenübertragung erfordern	Minimieren der Größe ausführbarer Programme, die keine Unterstützung für Datenübertragung erfordern
MOUSE.COM	Maustreiber zur Verwendung mit QuickBASIC	Verwendung der Maus
DEMO1.BAS, DEMO2.BAS, DEMO3.BAS	Beispielprogramme	Verwendet für die praktische Übung zu QuickBASIC in Kapitel 2, "Ihre erste QuickBASIC-Übung"
QB.HLP	Texte zur direkten (on-line) Hilfe	Direkte Hilfe

1.3 QuickBASIC installieren: Auf Festplatte

Die Datei SETUP.BAT, zu finden auf der QuickBASIC-Originaldiskette 1, kopiert QuickBASIC auf Ihre Festplatte.

Sie können Ihre QuickBASIC-Dateien auch installieren, ohne SETUP.BAT zu verwenden. Abhängig davon, wie Sie QuickBASIC verwenden möchten, müssen Sie die entsprechenden Dateien aus obiger Tabelle installieren.

Sobald Sie die benötigten Dateien festgelegt haben, führen Sie einen der folgenden Schritte aus:

- Kopieren Sie all diese Dateien auf ein Verzeichnis Ihrer Festplatte.
- Kopieren Sie verschiedene Dateien auf verschiedene Verzeichnisse, und setzen Sie DOS-"Umgebungsvariablen", die sowohl DOS als auch dem Compiler mitteilen, wo die Dateien zu finden sind.

Die zweite Methode hält die Organisation Ihrer Festplatte übersichtlicher. Weitere Informationen finden Sie in Abschnitt 1.7, "Wie Sie DOS-Umgebungsvariablen setzen".

1.4 QuickBASIC installieren: Auf Diskette

Um QuickBASIC auf einem Diskettensystem zu installieren, verwenden Sie den DOS-Befehl COPY, um die Dateien von den Originaldisketten auf Ihre eigenen Disketten zu kopieren. Um die von Ihnen benötigten Dateien zu bestimmen, lesen Sie in obiger Tabelle nach.

Die empfohlene Disketten-Konfiguration zur Verwendung bei einem System mit zwei Disketten-Laufwerken beschreibt die folgende Tabelle. Sie benötigen vier formatierte Disketten.

Nummer der Diskette	*Dateien*
1	COMMAND.COM
	AUTOEXEC.BAT (wahlfrei)
	CONFIG.SYS (wahlfrei)
	QB.EXE
	QB.HLP
	MOUSE.COM
2	BRUN40.EXE
	BRUN40.LIB
	BQLB40.LIB
	LINK.EXE
	LIB.EXE
	BC.EXE

Nummer der Diskette	*Dateien*
3	BCOM40.LIB
	LINK.EXE
	QB.LIB
	QB.BI*
4	BASIC Quelldateien
	QB.QLB (wahlfrei)**
	Quick-Bibliotheken

* Diese Datei ist nur dann erforderlich, wenn Ihr Programm QB.LIB zur Unterstützung von DOS-Interrupts verwendet. Weitere Informationen finden Sie in Kapitel 8, "Quick-Bibliotheken".

** Diese Quick-Bibliothek ist nur dann erforderlich, wenn Ihr Programm DOS-Interrupts verwendet. Weitere Informationen finden Sie in Kapitel 8, "Quick-Bibliotheken".

Unter Verwendung der obigen Disketten-Konfiguration beachten Sie bitte, daß Sie QuickBASIC von Laufwerk B aus starten, während QB.EXE in Laufwerk A ist. Es ist damit sichergestellt, daß Ihr Programm in der Lage ist, jede von ihm benötigte Bibliothek zu finden, und ausführbare Dateien werden auf die Diskette im Laufwerk B geschrieben. Um QuickBASIC von Laufwerk B aus zu starten, legen Sie Diskette 1 in Laufwerk A und Diskette 4 in Laufwerk B ein.

1. Tippen Sie `b:` und betätigen anschließend die EINGABETASTE, um Laufwerk B zu aktivieren.
2. Tippen Sie `a:qb`, um QuickBASIC zu starten.

Meistens werden Sie wahrscheinlich nur Diskette 1 sowie Diskette 4 benötigen. Die Disketten 2 und 3 werden Sie in einigen der folgenden Fälle benötigen:

Zu bearbeitende Aufgabe	*Laufwerks-Konfiguration*
Programme ohne eine Quick-Bibliothek in den Speicher schreiben und im Speicher ausführen	Legen Sie Diskette 1 in Laufwerk A und Diskette 4 in Laufwerk B ein.
Programme mit einer Quick-Bibliothek in den Speicher schreiben und im Speicher ausführen	Legen Sie Diskette 1 in Laufwerk A und Diskette 4 in Laufwerk B ein. Diskette 4 muß die BASIC-Quelldatei und die von dem Programm erforderte Quick-Bibliothek enthalten.

Zu bearbeitende Aufgabe	*Laufwerks-Konfiguration*
Erstellen einer Quick-Bibliothek oder eines ausführbaren Programmes, in dem die Option "EXE benötigt BRUN40.EXE" des Befehls **EXE-Datei erstellen** aus dem Menü **Ausführen** verwendet wird	Legen Sie Diskette 1 in Laufwerk A und Diskette 4 in Laufwerk B ein. Wenn Sie die Anfrage `Datei nicht gefunden:` *Dateiname* sehen, legen Sie Diskette 2 in Laufwerk A ein, und betätigen Sie die EINGABETASTE. Wenn Ihr Programm eine Quick-Bibliothek benötigt, stellen Sie sicher, daß die Bibliothek sich auf der Diskette in Laufwerk B befindet.
Erstellen eines ausführbaren Programmes, in dem die Option "Selbständige EXE-Datei" des Befehls **EXE-Datei erstellen** aus dem Menü **Ausführen** verwendet wird	Legen Sie Diskette 1 in Laufwerk A und Diskette 4 in Laufwerk B ein. Wenn Sie die Anfrage `Datei nicht gefunden:` *Dateiname* sehen, tippen Sie `a:`, legen die entsprechende Diskette in Laufwerk A ein und betätigen die EINGABETASTE. (Bei der ersten Anfrage wird dies die Diskette 2 mit den Dateien BC.EXE und LINK.EXE sein; bei der zweiten Anfrage wird dies die Diskette 3 sein, die BCOM40.LIB enthält.) Schließlich werden Sie aufgefordert werden, die Diskette mit der Datei QB.EXE einzulegen.

Sie werden nicht in der Lage sein, eine allein ausführbare Datei von einem Programm aus zu erzeugen, das die Quick-Bibliothek benützt. Wenn Sie auf ein derartiges Problem treffen, erzeugen Sie die allein ausführbare Datei, indem Sie die Original-Module in QuickBASIC laden.

Hinweis Wenn Sie ein System mit nur einem Laufwerk oder 3,5"-Disketten verwenden, finden Sie in der Datei INFO.TXT auf der Originaldiskette 1 einen Vorschlag zur Disketten-Installation.

1.5 Wenn Sie mit einer Maus arbeiten

QuickBASIC können Sie mit oder ohne Maus verwenden. Darüber hinaus können Sie eine Kombination von Maustechniken mit Tastaturbefehlen benutzen.

Microsoft QuickBASIC ist für die Benutzung mit der Microsoft-Mouse entwickelt worden. Obwohl viele Hersteller Ihre Zeigegeräte als kompatibel zu der Microsoft-Mouse anpreisen, arbeitet QuickBASIC nur dann sicher mit diesen, wenn sie die Funktionsaufrufe der Microsoft-Mouse exakt emulieren. Setzen Sie sich bitte hinsichtlich der Einzelheiten mit dem jeweiligen Hersteller in Verbindung.

Sie können QuickBASIC mit jeder beliebigen Version des Maustreibers MOUSE.SYS verwenden. Wenn Ihr Programm jedoch Funktionsaufrufe der Maus benutzt, müssen Sie MOUSE.COM, Version 6.11, einsetzen, die auf der QuickBASIC-Originaldiskette 3 geliefert wird. Das Programm SETUP kopiert diese Datei automatisch auf Ihre Festplatte. Wenn Sie sich entschließen, SETUP nicht zu benutzen, aber mit einer Maus arbeiten, sollten Sie die mit diesem Produkt gelieferte Version von MOUSE.COM kopieren.

1.6 Wenn Sie einen mathematischen Koprozessor einsetzen

Die mathematischen Koprozessoren 8087 und 80287 unterstützen eine schnelle und genaue "Gleitkomma-Berechnung". Gleitkomma-Berechnung umfaßt Arithmetik, die mit Zahlen durchgeführt wird, die keine ganzen Zahlen sind: 4,567/(23,4^2) ist ein Beispiel für eine Gleitkomma-Berechnung.

Microsoft QuickBASIC verwendet einen dieser mathematischen Koprozessoren automatisch, wenn ein solcher vorhanden ist. Andernfalls benutzt QuickBASIC Software-Routinen, um einen Koprozessor zu emulieren. Das Emulieren des Koprozessors führt zu einer langsameren Ausführung, als mit einem Koprozessor erreicht werden kann, bietet jedoch einen ähnlichen Grad an Genauigkeit.

Hinweis Es ist möglich, QuickBASIC zu veranlassen, das Vorhandensein eines 8087- oder 80287-Koprozessors zu ignorieren, indem die Umgebungsvariable NO87 gesetzt wird. Weitere Informationen finden Sie in Abschnitt 1.7.3, "Die Umgebungsvariable NO87".

Die Benutzung eines Koprozessors – oder dessen Emulation per Software – läßt Gleitkomma-Arithmetik generell viel genauer werden als mit BASICA oder früheren Versionen von QuickBASIC.

1.7 Wie Sie DOS-Umgebungsvariablen setzen

Wenn Sie nicht alle Ihre QuickBASIC-Dateien in einem Verzeichnis unterbringen möchten, können Sie die Dateien in verschiedenen Verzeichnissen ablegen, anschließend die DOS-Befehle PATH und SET verwenden, um Umgebungsvariablen zu definieren, die QuickBASIC mitteilen, wo die benötigten Dateien zu finden sind.

Sie müssen zwar keine Umgebungsvariable zur Ausführung von QuickBASIC setzen, aber Sie werden feststellen, daß deren Verwendung hilfreich ist. Falls Sie keine Umgebungsvariablen setzen, durchsucht QuickBASIC nur das aktuelle Arbeitsverzeichnis nach benötigten Dateien und gibt die Meldung `Datei nicht gefunden` *Datei*. `Pfad eingeben:` aus, wenn es diese nicht finden kann.

Es folgen einige Dateien, die QuickBASIC benötigt:

Dateityp und -erweiterung	***Beschreibung***
Ausführbare Dateien (.EXE)	Dateien, die ausgeführt werden, während Ihr Programm verarbeitet wird. Diese umfassen QB.EXE (wenn Sie innerhalb der Programmierumgebung arbeiten) und – wenn Sie selbständige Programme kompilieren und binden oder Bibliotheken erstellen – den BC.EXE-Compiler, den LINK.EXE-Linker, das BRUN40.EXE-Laufzeitmodul und den LIB.EXE-Bibliotheksmanager.
Bibliotheksdateien (.LIB)	Die Bibliotheksdateien BRUN40.LIB, BCOM40.LIB und BQLB40.LIB, die der Linker LINK.EXE erfordert.

Wenn Sie eine Umgebungsvariable einmal gesetzt haben, bleibt diese solange wirksam, bis Sie die Umgebungsvariable erneut mit einem anderen Wert setzen, Ihren Computer erneut starten oder ausschalten.

Die Abschnitte 1.7.1 und 1.7.2 beschreiben kurz zwei der am häufigsten verwendeten Umgebungsvariablen – PATH und LIB – und erläutern, wie diesen Werte zugewiesen werden. Wenn Sie einen 8087 oder einen 80287 mathematischen Koprozessor haben, sollten Sie zum Setzen der Umgebungsvariablen NO87 auch Abschnitt 1.7.3 lesen, wenn Sie wünschen, daß Ihre Programme einen mathematischen Koprozessor ignorieren.

1.7.1 Die Umgebungsvariable PATH

Die Umgebungsvariable PATH teilt dem Compiler mit, wo er nach ausführbaren Dateien suchen soll. Sowohl QB.EXE als auch BC.EXE suchen nach der Umgebungsvariablen PATH.

Verwenden Sie den DOS-Befehl PATH, um die Variable PATH zu definieren. Geben Sie den Befehl, ein Leerzeichen oder ein Gleichheitszeichen und eine oder mehrere durch Semikolons (;) getrennte Pfadangaben ein. Die folgende Vereinbarung teilt dem Compiler zum Beispiel mit, ausführbare Dateien im Laufwerk C zu suchen, zunächst in dem Verzeichnis \BIN, anschließend in dem Verzeichnis \QB:

```
PATH=C:\BIN;C:\QB
```

Mit der in diesem Beispiel gezeigten Vereinbarung für PATH könnten Sie QB.EXE in das Verzeichnis \BIN schreiben und anschließend QuickBASIC aus einem beliebigen Verzeichnis starten.

Hinweis Wenn Sie mit einer DOS-Version 3.0 oder größer arbeiten, können Sie die Variable PATH mit dem Befehl SET definieren, wie in der nächsten Zeile gezeigt:

```
SET PATH=C:\BIN;C:\QB
```

Wenn Sie mit einer DOS-Version kleiner Version 3.0 arbeiten, können Sie die Variable PATH nicht mit dem Befehl SET definieren. Mit diesen früheren Versionen *müssen* Sie den Befehl PATH verwenden.

1.7.2 Die Umgebungsvariable LIB

Die Umgebungsvariable LIB teilt QuickBASIC mit, wo eine auf der **qb**-Befehlszeile angegebene Quick-Bibliothek ausfindig zu machen ist, und teilt LINK.EXE mit, wo es alle die von ihm benötigten Bibliotheksdateien findet. Definieren Sie die Umgebungsvariable LIB mit dem Befehl SET. Der folgende Befehl teilt dem Linker zum Beispiel mit, nach Bibliotheken in dem Verzeichnis \LIB im Laufwerk C zu suchen:

```
SET LIB=C:\LIB
```

1.7.3 Die Umgebungsvariable NO87

Wie bereits in Abschnitt 1.6 erwähnt, werden Ihre Programme während der Laufzeit einen 8087/80287-Koprozessor verwenden, wenn ein solcher installiert ist. Sie können die Verwendung des Koprozessors jedoch unterdrücken und das Programm dazu veranlassen, die Funktion des Koprozessors zu emulieren, indem Sie die Umgebungsvariable NO87 setzen. Die folgenden Zeilen zeigen zwei Möglichkeiten, die Variable NO87 zu setzen:

`SET NO87=Verwendung des Koprozessors unterdrückt`

`SET NO87=`*ein oder mehrere Leerzeichen*

Die erste Vereinbarung führt zu der Meldung `Verwendung des Koprozessors unterdrückt` auf dem Bildschirm, wenn ein Programm ausgeführt wird, das einen 8087 oder 80287 verwendet, und ein 8087 oder 80287 vorhanden ist. Die zweite Vereinbarung gibt keine Meldung aus.

Um die erzwungene Emulation eines Koprozessors auszuschalten, setzen Sie NO87 gleich einem Null- oder leeren Wert, wie im nächsten Beispiel:

```
SET NO87=
```

Beachten Sie, daß dem Gleichheitszeichen in diesem Beispiel keine Leerzeichen folgen.

2 Ihre erste QuickBASIC-Übung

Dieses Kapitel bietet Ihnen, abhängig von Ihrer früheren Programmiererfahrung mit BASIC, drei Möglichkeiten für die ersten Schritte in QuickBASIC:

Ihre Erfahrung	*Abschnitte, die Sie benutzen sollten*
Keine BASIC-Programmiererfahrung oder Programmierung in BASICA oder anderem interpretierten BASIC, nie zuvor jedoch QuickBASIC eingesetzt	Abschnitt 2.1 "Wenn Sie BASICA-Programmierer sind", sowie Abschnitt 2.4, "Grundlegende Informationen für alle QuickBASIC-Benutzer"
Programmierung in anderen QuickBASIC-Versionen	Abschnitt 2.2, "Wenn Sie mit QuickBASIC, Version 2.0 oder 3.0, gearbeitet haben" und Abschnitt 2.4, "Grundlegende Informationen für alle QuickBASIC-Benutzer"
Programmierung mit einem Microsoft BASIC-Befehlszeilen-Compiler	Abschnitt 2.3, "Wenn Sie mit einem BASIC-Compiler gearbeitet haben" und Abschnitt 2.4, "Grundlegende Informationen für alle QuickBASIC-Benutzer"

Wenn Sie dieses Kapitel beendet haben, werden Sie wissen, wie mit QuickBASIC die folgenden Aufgaben erledigt werden:

- QuickBASIC starten.
- Ein QuickBASIC-Programm laden.
- Ein QuickBASIC-Programm im Speicher ausführen.
- Eine "ausführbare" Datei (eine Datei mit der Erweiterung .EXE) auf Diskette erstellen.
- QuickBASIC beenden.

2.1 Wenn Sie BASICA-Programmierer sind

Beginnen Sie hier, wenn Sie noch niemals in BASIC programmiert haben, oder wenn Sie Programmierer für interpretiertes BASIC oder BASICA sind, der nie zuvor mit QuickBASIC gearbeitet hat. Wenn Sie bereits Erfahrungen mit anderen QuickBASIC-Versionen haben, fahren Sie mit Abschnitt 2.2 fort. Wenn sich Ihre Erfahrungen überwiegend auf einen Microsoft BASIC-Befehlszeilen-Compiler beziehen, können Sie diesen und den nächsten Abschnitt als eine Einführung in die Programm-Entwicklungsumgebung von QuickBASIC lesen oder direkt mit Abschnitt 2.3 fortfahren.

Bevor Sie mit den Übungen dieses Kapitels beginnen, sollten Sie QuickBASIC entsprechend der Hinweise in Kapitel 1 installiert haben. Außerdem benötigen Sie ein BASIC-Programm, mit dem Sie arbeiten. Beginnen Sie mit einem Ihrer eigenen BASIC-Programme oder dem Programm DEMO1.BAS, das auf der QuickBASIC-Originaldiskette 3 geliefert wird.

Versuchen Sie zunächst ein sehr einfaches Programm. Da sich einige Anweisungen und Funktionen in QuickBASIC etwas anders verhalten als in BASICA, sollten Sie, bevor Sie fortfahren, in Anhang A, "Übertragen von BASICA-Programmen nach QuickBASIC", nachschlagen, wenn Sie Ihre eigenen Programme verwenden. Das von Ihnen ausgewählte Programm sollte eine Ausgabe vornehmen, so daß Sie feststellen können, ob es korrekt läuft. Bis Sie umfassend mit dem QuickBASIC-Editor vertraut sind, sollten Sie eine Sicherungskopie jedes alten Programmes, das Sie in QuickBASIC laden, verwenden.

Wichtig Wenn Sie Ihre eigenen BASICA-Programme benutzen, müssen Sie sicherstellen, daß diese im ASCII-Format gespeichert sind. QuickBASIC kann BASICA-Programme, die im Binärformat vorliegen, nicht laden.

Wenn Sie nicht sicher sind, welches Format Ihr BASICA-Programm hat, schauen Sie es sich mit dem DOS-Befehl TYPE an. Wenn die Ausgabe des Befehls TYPE genauso aussieht wie das, was Sie sehen, wenn Sie Ihr Programm in BASICA auflisten, dann ist das Programm im ASCII-Format gespeichert. Wenn jedoch viele nichtalphabetische Zeichen auf dem Bildschirm erscheinen, dann ist das Programm im Binärformat gespeichert.

Wenn das Programm im Binärformat vorliegt, müssen Sie diese drei Schritte vor der Ausführung mit QuickBASIC durchführen:

1. Starten Sie BASICA mit der Eingabe:

   ```
   basica
   ```

2. Laden Sie Ihr Programm (BEISP.BAS in diesem Beispiel) mit der Eingabe:

   ```
   load "beisp"
   ```

3. Speichern Sie das Programm im ASCII-Format ab mit der Eingabe:

   ```
   save "beisp",a
   ```

4. Verlassen Sie BASICA mit der Eingabe:

   ```
   system
   ```

2.1.1 Wie Sie QuickBASIC starten

Wenn Sie einen Farbmonitor oder einen monochromen Monitor mit einem monochromen Adapter verwenden, starten Sie QuickBASIC mit der Eingabe

```
qb
```

und betätigen Sie die EINGABETASTE.

QuickBASIC kann den Typ der Bildschirm-Adapterkarte feststellen, nicht aber den Typ des Monitors. Wenn Sie einen Schwarzweiß(Composite)-Monitor mit einer CGA verwenden, starten Sie QuickBASIC mit der Eingabe

```
qb /b
```

und betätigen Sie die EINGABETASTE.

Wenn QuickBASIC nicht startet, stellen Sie sicher, daß sich die Datei QB.EXE in Ihrem aktuellen Verzeichnis oder in einem von Ihnen mit der PATH-Variablen festgelegten Verzeichnis befindet, und versuchen Sie es erneut.

Wenn QuickBASIC startet, sind Sie in der Lage, Ihr erstes Programm zu laden.

2.1.2 Wie Sie ein Programm laden

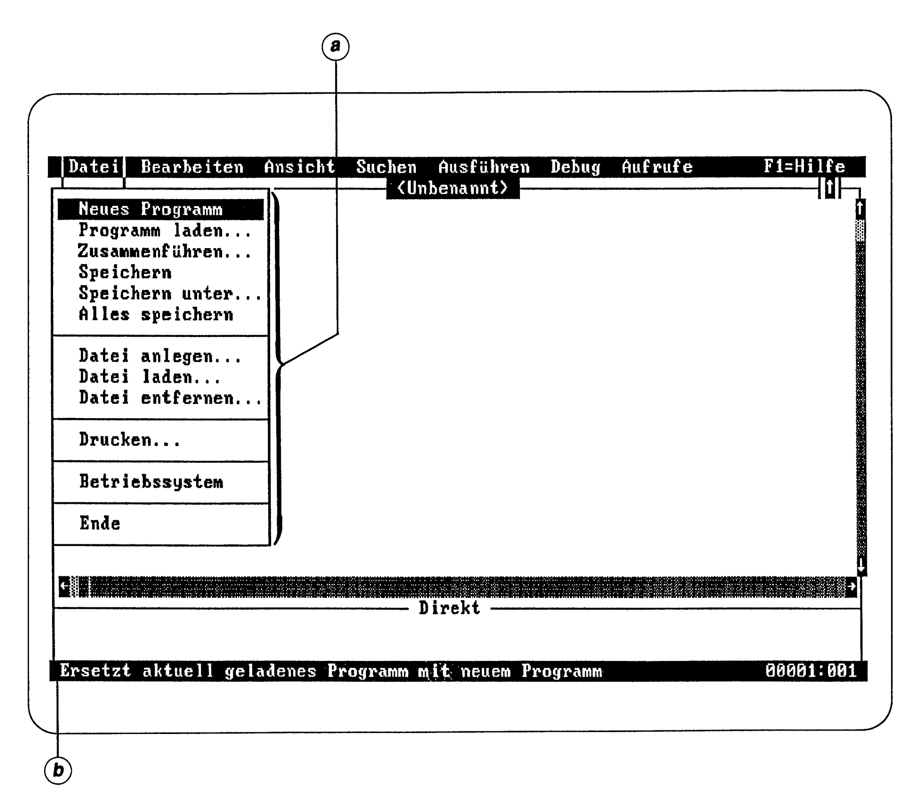
Bevor Sie ein Programm ausführen können, müssen Sie es in QuickBASIC laden. Folgen Sie diesen Schritten, um ein Programm zu laden:

1. Betätigen Sie die ALT-TASTE. Das Wort **Datei** oben links auf dem Bildschirm ist nun "markiert" (hervorgehoben).

   Beachten Sie, daß Sie die ALT-TASTE nicht gedrückt halten müssen. Wenn Sie mit Menüs arbeiten, können Sie die ALT-TASTE betätigen und loslassen, bevor Sie mit Schritt 2 fortfahren.

2. Betätigen Sie D, um die gesamte Liste der Befehle des Menüs **Datei** anzuzeigen (gezeigt in Abbildung 2.1). Eine vollständige Erläuterung der Menüs finden Sie in Kapitel 3, "Wie Sie sich in der QuickBASIC-Umgebung zurechtfinden".

Abbildung 2.1 Das Menü ***Datei***



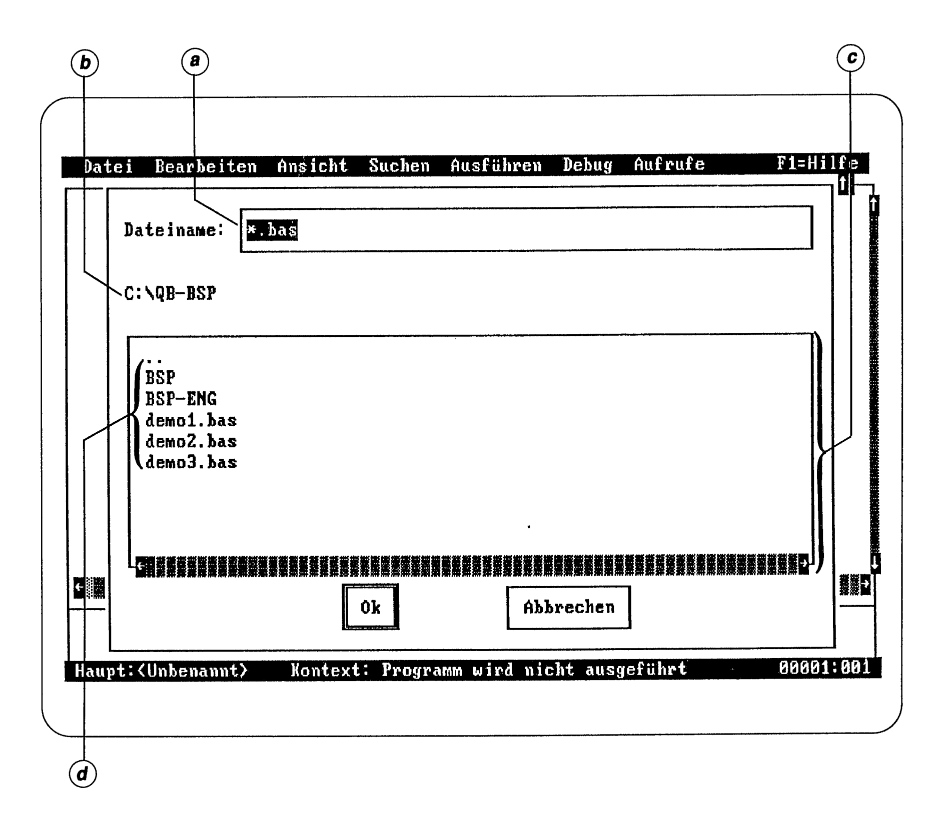
a) Menü ***Datei***

b) Die Hilfezeile gibt Informationen zu dem hervorgehobenen Befehl.

3. Betätigen Sie P, um aus dem Menü **Datei** den Befehl **Programm laden** auszuwählen.

   Es erscheint das in Abbildung 2.2 gezeigte "Dialogfeld" **Programm laden.** Ein Dialogfeld ist das Feld, in das Sie die von einem Befehl benötigten Informationen eingeben. In diesem Fall benötigt der Befehl **Programm laden** den Namen der Datei, die Sie laden möchten.

Abbildung 2.2 Dialogfeld ***Programm laden***



a) Geben Sie den Namen der Datei ein, die Sie laden möchten.
b) Verzeichnis der gezeigten Dateien.
c) Oder verwenden Sie die TAB-TASTE, um in das Listenfeld zu gelangen, und markieren Sie anschließend den Dateinamen.
d) Liste der Unterverzeichnisse und .BAS-Dateien im aktuellen Verzeichnis.

Ihr Bildschirm wird wahrscheinlich anders aussehen als der in Abbildung 2.2, da Sie sicherlich andere Dateien und Verzeichnisse auf Ihrer Diskette haben werden. Weitere Informationen zu dem Dialogfeld **Programm laden** finden Sie in Kapitel 4, "Wie Sie Quelldateien verwalten".

4. Geben Sie den Namen der Datei, die Sie laden möchten, ein (sowie einen wahlfreien Verzeichnispfad zu der Datei, wenn diese sich in einem anderen Verzeichnis befindet), und betätigen Sie die EINGABETASTE.

   Der von Ihnen eingegebene Name erscheint in dem Feld neben dem Wort Dateiname.

5. Wenn Sie eines Ihrer eigenen BASIC-Programme laden, können Sie eventuell eine Fehlermeldung sehen. Die wahrscheinlichste Ursache ist, daß Ihr Programm im Binärformat gespeichert wurde (das Sie erhalten, wenn Sie in BASICA nur

   ```
   save programmname
   ```

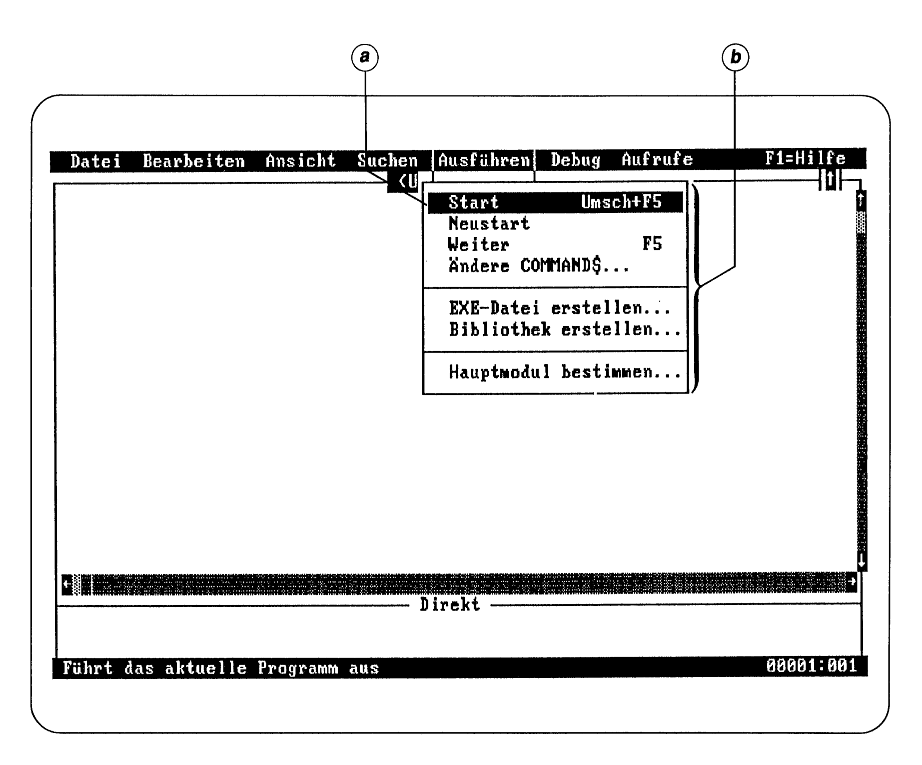
   eingeben; wie dieses Problem korrigiert werden kann, finden Sie in dem Hinweis "Wichtig" am Beginn dieses Abschnittes). Wenn Sie eine Fehlermeldung sehen, betätigen Sie die ESC-TASTE, um die Fehlermeldung von Ihrem Bildschirm zu löschen; wiederholen Sie dann die Schritte 1 bis 4, laden Sie aber diesmal das Programm DEMO1.BAS, das auf der QuickBASIC-Originaldiskette 3 geliefert wird.

2.1.3 Wie Sie Ihr Programm ausführen

Nachdem Sie nun Ihr Programm erfolgreich geladen haben, sind Sie bereit, es zu starten. Dieser Abschnitt zeigt, wie Sie Ihr Programm innerhalb der QuickBASIC-Umgebung ausführen. Die Verarbeitung, die QuickBASIC verwendet – auch als "Übersetzung in ausführbaren Code" bezeichnet – ist der Art ähnlich, mit der Sie Programme in BASICA ausführen. Folgen Sie diesen Schritten, um das Programm in QuickBASIC auszuführen:

1. Betätigen Sie die ALT-TASTE und anschließend A, um das in Abbildung 2.3 gezeigte Menü **Ausführen** zu öffnen.

*Abbildung 2.3 Das Menü **Ausführen***



*a) Wählen Sie **Start**, um das Programm zu starten.*
*b) Menü **Ausführen***

2. Betätigen Sie S, um den Befehl **Start** zu wählen. Dieser Befehl startet das Programm.

Wenn Sie eines Ihrer eigenen BASICA-Programme starten, erhalten Sie vielleicht einen Fehler. Einige Anweisungen erfordern Veränderungen, um in QuickBASIC richtig ausgeführt zu werden. Diese Anweisungen sind in Anhang A, "Übertragen von BASICA-Programmen nach QuickBASIC" beschrieben. Wenn Sie eine Fehlermeldung sehen, folgen Sie diesen Schritten, um das aktuelle Programm durch das Beispielprogramm DEMO1.BAS, das auf der Originaldiskette 3 geliefert wird, zu ersetzen:

1. Betätigen Sie die ESC-TASTE, um die Fehlermeldung zu löschen.

2. Betätigen Sie die ALT-TASTE und anschließend D, um das Menü **Datei** zu öffnen.

3. Betätigen Sie P, um das Dialogfeld **Programm laden** anzuzeigen.

4. Geben Sie

```
DEMO1.BAS
```

ein, und betätigen Sie die EINGABETASTE, um das Programm zu laden.

5. Wiederholen Sie die Schritte der vorhergehenden Auflistung, um DEMO1.BAS zu starten.

Hinweis Sie können Ihr Programm in eine "selbständig ausführbare Datei" umwandeln, indem Sie eine als "Kompilieren auf Diskette" bezeichnete Verarbeitung verwenden. Ein selbständiges Programm, wie der Name schon sagt, erfordert es nicht, daß Sie vor der Ausführung des Programms QuickBASIC laden. Diese Fähigkeit, selbständige Programme zu erzeugen, ist einer von vielen Vorteilen, die QuickBASIC gegenüber BASICA hat. Weitere Informationen finden Sie in Abschnitt 2.1.5, "Erstellen eines ausführbaren Programms auf Diskette".

2.1.4 Ein Vergleich zwischen Interpretieren, Kompilieren und Ausführen in QuickBASIC

Sowohl Interpreter als auch Compiler sind Programme, die die von Ihnen geschriebenen Anweisungen in Befehle übersetzen, die von dem Computer verstanden werden können. Ein traditioneller Interpreter übersetzt und führt jede Anweisung aus, bevor er zur nächsten Anweisung übergeht, wogegen traditionelle Compiler alle Anweisungen eines Programms übersetzen, bevor irgendeine Ausführung beginnt. Viele Compiler erfordern darüber hinaus weitere Schritte, die der Ausführung eines Programmes vorangehen.

BASICA ist ein Beispiel für einen "Interpreter". Wenn Sie ein Programm in BASICA laden und dieses ausführen, wird jede Anweisung übersetzt und anschließend ausgeführt, bevor BASICA dazu übergeht, die nächste Anweisung zu übersetzen und auszuführen. Diese Verarbeitung ist ähnlich der eines menschlichen Dolmetschers, der Ihnen hilft, ein in einer fremden Sprache geschriebenes Buch zu verstehen. Jedesmal, wenn Sie einen Satz des Buches verstehen müssen, steht Ihnen der Dolmetscher zur Übersetzung des Satzes zur Verfügung. Ein Nachteil dieser Verarbeitung ist offensichtlich: Sie benötigen immer den Dolmetscher, um Dinge für Sie zu entziffern.

Das QuickBASIC-Paket umfaßt zwei verschiedene Werkzeuge für die Übersetzung Ihrer Programme. Dasjenige, das Sie, wie in den vorhergehenden Abschnitten beschrieben, eingesetzt haben, ist die Programm-Entwicklungsumgebung QB.EXE. Das andere, als BC.EXE bezeichnet, ist ein Befehlszeilen-Compiler. Es ist nur ein Übersetzer - es führt keine Programme aus. In diesem Sinne ist die Verwendung von BC gleichbedeutend damit, daß der oben erwähnte menschliche Dolmetscher eine Übersezung in Ihre Sprache für das *gesamte* Buch schreibt. Nachdem dies einmal geschehen ist, benötigen Sie nicht länger die Dienste des Dolmetschers; stattdessen können Sie in der Übersetzung nachschlagen.

Ein mit BC.EXE kompiliertes Programm kann als eine Diskettendatei, die als selbständig ausführbares Programm bezeichnet wird, gebunden und gespeichert werden. (Dateien von selbständig ausführbaren Programmen bestehen aus dem Basisnamen der ursprünglichen BASIC-Quelldatei und der Erweiterung .EXE und werden als "ausführbare" Dateien bezeichnet.) Darüber hinaus laufen kompilierte Programme viel schneller als interpretierte Programme, da ein Compiler Ihr Programm nur einmal übersetzt (während ein Interpreter jede Anweisung bei jeder erneuten Ausführung abermals übersetzen muß).

QB.EXE, die Programm-Entwicklungsumgebung von QuickBASIC, ist weder ein traditioneller Interpreter noch Compiler, obwohl sie das Beste beider Welten kombiniert: den Komfort eines Interpreters und die Ausführungsgeschwindigkeit eines Compilers. Sie läßt Ihnen die Wahl zwischen der Arbeit mit Ihrem Programm im Speicher (so daß Sie dieses jederzeit starten können) oder der Kompilierung des Programms und dessen Abspeicherung als eine Diskettendatei, die Sie außerhalb der Umgebung ausführen können. Wenn Sie mit einem Programm im Speicher arbeiten, übersetzt QuickBASIC jede Zeile während der Eingabe (anders als traditionelle Compiler, die die Übersetzung als völlig eigenständigen Schritt durchführen). Daher ist Ihr Programm in QuickBASIC jederzeit bereit, ausgeführt zu werden, unabhängig vom Grad der Fertigstellung. Weitere Informationen finden Sie in Abschnitt 2.4.1.1, "Die Entwicklungsumgebung, die Sie sich immer gewünscht haben".

Das Arbeiten mit Programmen im Speicher ist hilfreich während der Programmentwicklung, da Sie das Programm leicht ändern und Fehler beseitigen sowie anschließend das Programm sofort erneut starten können, um Ihre Änderungen zu überprüfen. Nachdem Sie Ihr Programm einmal komplett fehlerfrei vorliegen haben, können Sie eine ausführbare Datei erstellen und das Programm direkt aus DOS heraus starten, ohne zunächst QuickBASIC starten zu müssen.

Der nächste Abschnitt beschreibt den Ablauf zur Erstellung ausführbarer Dateien mit QuickBASIC.

2.1.5 Erstellen eines ausführbaren Programms auf Diskette

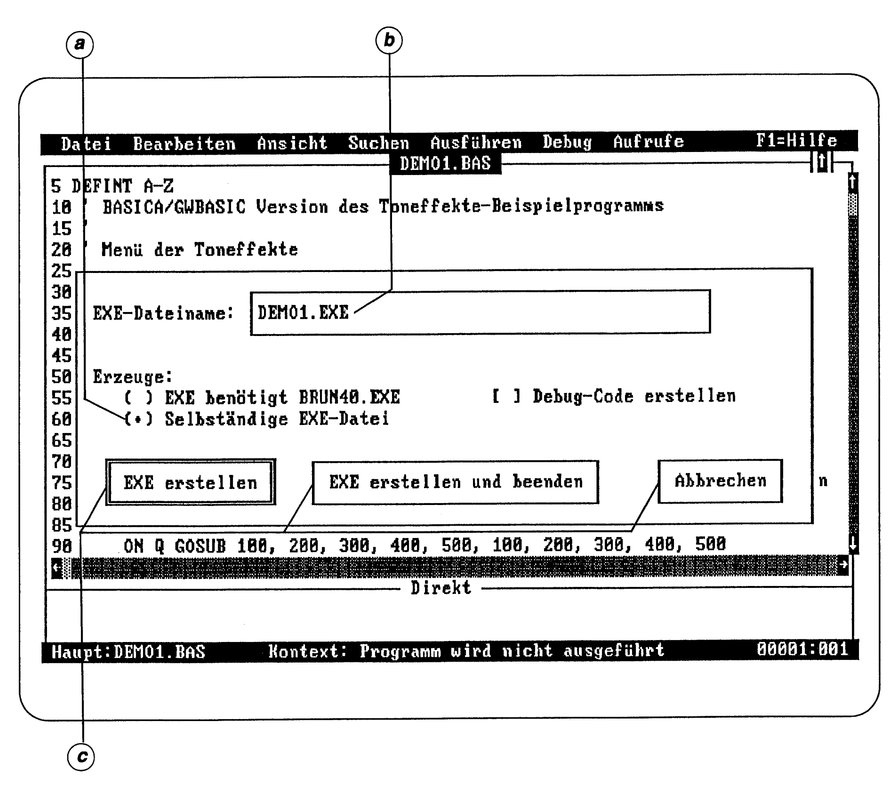
Nachdem Sie ein Programm innerhalb von QuickBASIC erfolgreich geladen und ausgeführt haben, können Sie eine ausführbare Datei auf Diskette erzeugen.

Folgen Sie diesen Schritten, um in dem aktuellen Verzeichnis eine ausführbare Datei zu erstellen:

1. Betätigen Sie die ALT-TASTE und anschließend A, um das Menü **Ausführen** zu öffnen.
2. Betätigen Sie E, um den Befehl **EXE-Datei erstellen** zu wählen. Es erscheint das in Abbildung 2.4 gezeigte Dialogfeld. Der Name der ausführbaren Datei erscheint in dem Feld neben dem Wort "EXE-Dateiname".



Abbildung 2.4 Das Dialogfeld ***EXE-Datei erstellen***



a) Wählen Sie aus diesen beiden Optionen die Option "Selbständige EXE-Datei".

b) Geben Sie den Namen der zu erstellenden ausführbaren Datei ein.

c) Befehlsflächen führen die gewählten Optionen aus oder brechen die gewählten Optionen ab.

3. Betätigen Sie die TAB-TASTE und anschließend S, um die Option "Selbständige EXE-Datei" zu wählen, da Sie den Dateinamen nicht verändern möchten.

4. Betätigen Sie E, um die Befehlsfläche "EXE erstellen" zu wählen.

Während das Programm kompiliert, werden Sie Meldungen des BC-Compilers und des Linkers sehen. Wenn diese Verarbeitung abgeschlossen ist, befinden Sie sich wieder in der QuickBASIC-Umgebung.

2.1.6 Wie Sie QuickBASIC verlassen

Um die selbständig ausführbare Datei zu testen, müssen Sie QuickBASIC verlassen und zu DOS zurückkehren. Um QuickBASIC zu verlassen, müssen Sie

1. die ALT-TASTE und anschließend D betätigen, um das Menü **Datei** anzuzeigen.
2. E betätigen, um den Befehl **Ende** zu wählen.

   Es erscheint die DOS-Eingabeaufforderung. (Wenn Sie an dem Programm Veränderungen vorgenommen haben, erscheint zunächst ein Dialogfeld, das Sie fragt, ob Sie die geladene Datei speichern möchten.)
3. J betätigen, um die Änderungen zu speichern, oder N, um QuickBASIC zu verlassen, ohne die Änderungen zu speichern.

2.1.7 Wie Sie ein Programm aus DOS heraus starten

Verwenden Sie den DOS-Befehl DIR, um sich den Namen der ausführbaren Datei, die Sie gerade erstellt haben, anzeigen zu lassen. Sie müßten eine Datei mit dem Basisnamen finden, der in dem Textfeld "EXE-Dateiname" erschien, einschließlich der Erweiterung .EXE.

Um die selbständige Version Ihres Programms zu starten

- geben Sie den Basisnamen der Datei ein, und betätigen Sie die EINGABETASTE.

2.1.8 Wie sich QuickBASIC von BASICA unterscheidet

QuickBASIC unterscheidet sich von BASICA sowohl in seiner "Programmierumgebung" – der Teil von QuickBASIC, in dem Sie Ihr Programm bearbeiten und debuggen – als auch durch die Erweiterungen der Sprache BASIC selbst. Beide Unterschiede werden kurz in den nächsten beiden Abschnitten und ausführlicher im Abschnitt 2.4.1, "Warum QuickBASIC besser ist als andere BASIC-Produkte", beschrieben.

2.1.8.1 Umgebungserweiterungen

Die Erweiterungen der QuickBASIC-Umgebung umfassen Änderungen der Methoden zum Bearbeiten und Debuggen von Dateien.

- **Bearbeiten**

  Der Mittelpunkt der QuickBASIC-Umgebung ist ihr "intelligenter" Editor. Obwohl dieser Editor demjenigen gleicht, den Sie mit BASICA verwendet haben, bietet er die folgenden Eigenschaften, die in BASICA nicht zu finden sind:

  - Überprüfung der Syntax während der Eingabe jeder Zeile eines Programms. Dies entdeckt die meisten syntaktischen Fehler sowie fehlende Satzzeichen (wie Kommata oder Hochkommata), bevor Sie Ihr Programm starten.
  - Automatische Großschreibung von Schlüsselwörtern, wie zum Beispiel **PRINT**. Dies hilft Ihnen, Schlüsselwörter von Ihren eigenen Marken und Namen zu unterscheiden und dient als weitere syntaktische Überprüfung, während Sie Ihr Programm eingeben.

  Eine vollständige Beschreibung des QuickBASIC-Editors finden Sie in Kapitel 5, "Bearbeiten".

- **Debuggen**

  Mit den Debug-Befehlen von QuickBASIC sind Sie in der Lage, Variablenwerte zu überprüfen und ein Programm bei jeder beliebigen Anweisung, oder wenn eine Variable einen vorbestimmten Wert erreicht, anzuhalten. Darüber hinaus können Sie in QuickBASIC ein Programm anhalten, Änderungen durchführen und anschließend an dem Punkt fortfahren, an dem Sie das Programm angehalten haben, wobei alle Daten unberührt bleiben. Wenn Sie eine Änderung durchgeführt haben, die das Programm am Fortfahren hindert, bietet QuickBASIC die Auswahl zwischen einem Fortfahren ohne die Änderung bzw. einem Start am Beginn des Programms mit der eingearbeiteten Änderung.

  Im Gegensatz dazu ist das Debuggen in BASICA begrenzt auf **TRON/TROFF** sowie den gezielten Einsatz der Anweisungen **PRINT**, **STOP** und **CONT**. Obwohl BASICA es Ihnen erlaubt, ein Programm anzuhalten und dessen Variablenwerte zu überprüfen, können Sie das Programm nicht verändern und an dem Punkt fortsetzen, an dem Sie es angehalten haben.

  Genauso wie BASICA unterstützt QuickBASIC Anweisungen im Direktmodus. In BASICA sind dies Anweisungen, die ohne Zeilennummern eingegeben werden. In QuickBASIC werden Anweisungen im Direktmodus in das Direkt-Fenster eingegeben.

2.1.8.2 Neue Sprachmerkmale

QuickBASIC erweitert die Sprache BASIC, um viele der Merkmale strukturierterer Sprachen wie Pascal und C bereitzustellen. Diese Erweiterungen umfassen folgendes:

- **SUB**- und **FUNCTION**-Prozeduren erleichtern Ihnen strukturierte Programmierung.
- **TYPE...END TYPE**-Anweisungen versetzen Sie in die Lage, unterschiedliche Variablentypen in einem zusammengesetzten benutzerdefinierten Typ zusammenzufassen. Dies vereinfacht Eingaben von bzw. Ausgaben auf Direktzugriffsdateien sehr.

- Neue Konstruktionen zur Ablaufsteuerung, die **SELECT CASE**, Block-**IF...THEN...ELSE** und **DO...LOOP** umfassen, sind hilfreich, die Logik Ihres Programmes eindeutig und leicht veränderbar zu gestalten.
- Lange (32-Bit-) Ganzzahlen ermöglichen es Ihnen, Finanzberechnungen mit Pfennigen durchzuführen, so daß die Ergebnisse genau sind (keine Rundungsfehler). Diese Berechnungen können schneller durchgeführt werden, als es der Fall ist, wenn für die Berechnungen mit DM und Pfenningen Zahlen einfacher Genauigkeit verwendet werden.
- Rekursion erlaubt es **SUB**- und **FUNCTION**-Prozeduren, sich selbst aufzurufen.
- Zahlen im IEEE-Format bieten zusätzlichen Umfang und zusätzliche Genauigkeit für Gleitkomma-Berechnungen.
- Binärer Zugriff auf Disketten-Dateien ermöglicht es Ihnen, Nicht-ASCII-Disketten-Dateien zu lesen und zu verändern.

2.1.9 Wie es weitergeht

Sie kennen nun die Grundlagen für die Benutzung von QuickBASIC: Programme in den Speicher laden und im Speicher ausführen sowie Programme in selbständig ausführbare Dateien kompilieren. Als nächstes möchten Sie wahrscheinlich mehr lernen über Menüs, Befehle und Dialogfelder; die neuen Merkmale, die QuickBASIC zu der Sprache BASIC hinzufügt; und wie Sie den Editor einsetzen. Die folgende Aufstellung zeigt Ihnen, wo entsprechende Informationen zu finden sind:

Information	*Fundstelle*
Weitere Informationen zu neuen Merkmalen	Abschnitt 2.4.1, "Warum QuickBASIC besser ist als andere BASIC-Produkte"
Beim Start von QuickBASIC zu verwendende Optionen	Abschnitt 2.4.2, "Der Befehl qb"
Die Verwendung von Menüs, Fenstern und Dialogfeldern	Kapitel 3, "Wie Sie sich in der QuickBASIC-Umgebung zurechtfinden"
Verändern Ihrer alten BASICA-Programme, damit diese mit QuickBASIC laufen	Anhang A, "Übertragen von BASICA-Programmen nach QuickBASIC"
Programmieren in BASIC mit neuen Sprachmerkmalen	*Programmieren in BASIC: Ausgewählte Themen*
Einrichten des QuickBASIC-Bildschirms	Abschnitt 2.4.3, "Einrichten des QuickBASIC-Bildschirms"
Das Hilfesystem von QuickBASIC benutzen	Abschnitt 2.4.5, "Wie Sie von QuickBASIC Hilfe erhalten"

2.2 Wenn Sie mit QuickBASIC, Version 2.0 oder 3.0, gearbeitet haben

Beginnen Sie hier, wenn Sie mit früheren Versionen von QuickBASIC gearbeitet haben. Wenn Ihre Erfahrungen sich auf die Verwendung eines BASIC-Interpreters wie BASICA beschränken (oder wenn Sie BASIC überhaupt noch nicht eingesetzt haben), sollten Sie mit Abschnitt 2.1 beginnen. Wenn Sie mit einem Microsoft BASIC-Befehlszeilen-Compiler gearbeitet haben und die Eigenschaften der QuickBASIC-Umgebung nicht verwenden möchten, fahren Sie mit Abschnitt 2.3 fort.

Bevor Sie den Rest dieses Abschnittes lesen, sollten Sie QuickBASIC entsprechend der Hinweise in Kapitel 1 installiert haben.

Hinweis Obwohl die meisten in früheren QuickBASIC-Versionen geschriebenen Programme sofort geladen und gestartet werden können, können bestimmte Unterschiede zwischen dieser und früheren QuickBASIC-Versionen leichte Veränderungen alter Programme erfordern. Die folgenden Abschnitte zeigen Ihnen wichtige Unterschiede in Sprache und Umgebung auf und bieten Ihnen die Informationen, die Sie benötigen, um Ihre Programme ausführen zu können. QuickBASIC behandelt einige Formatierungsaufgaben, insbesondere Tabulatoren setzen und Einrückungen, anders als andere Editoren.

Bis Sie mit dem QuickBASIC-Editor vollständig vertraut sind, sollten Sie mit Sicherungskopien Ihrer alten Quelldateien arbeiten.

Die Abschnitte 2.2.1 bis 2.2.4 zeigen Ihnen, wie Sie QuickBASIC starten, und führen Sie dann durch das Laden, die Überwachung und das Ausführen eines Programms. Die Abschnitte 2.2.5 bis 2.2.6 erläutern, wie ausführbare Dateien innerhalb der QuickBASIC-Umgebung erstellt und anschließend aus DOS heraus gestartet werden. Die Abschnitte 2.2.7 bis 2.2.8 beschreiben die Unterschiede zwischen dieser QuickBASIC-Version und anderen und weisen auf andere Teile dieses Handbuches hin, die hilfreich für Sie sein werden.

2.2.1 Wie Sie QuickBASIC starten

Das Starten dieser QuickBASIC-Version ist ähnlich dem Starten früherer Versionen. Wenn Sie einen Farbmonitor oder einen monochromen Monitor mit einem monochromen Grafikadapter einsetzen, geben Sie

```
qb
```

ein, und betätigen Sie die EINGABETASTE.

QuickBASIC kann den Typ der Bildschirm-Adapterkarte feststellen, nicht aber den Typ des Monitors. Wenn Sie einen Schwarzweiß(composite)-Monitor mit einer CGA einsetzen, geben Sie

```
qb /b
```

ein, und betätigen Sie die EINGABETASTE.

Wenn Sie eine Fehlermeldung erhalten, stellen Sie sicher, daß sich die Datei QB.EXE in Ihrem aktuellen Verzeichnis oder in einem von Ihnen mit der PATH-Variablen festgelegten Verzeichnis befindet, und versuchen Sie es erneut.

2.2.2 Wie Sie QuickBASIC-Programme laden

Programme, die mit QuickBASIC, Version 2.0 oder 3.0, erstellt wurden, sollten mit einer Ausnahme ohne Schwierigkeiten geladen werden können: wenn Sie Programme haben, die Include-Dateien verwenden, die **SUB...END SUB**-Anweisungen enthalten, sollten Sie vor dem Versuch, diese zu laden, Abschnitt 4.7, "Die Inhalte zweier Dateien zusammenführen", lesen. Für jetzt sollten Sie ein Programm wählen, das keine **SUB...END SUB**-Anweisungen enthaltende Include-Dateien verwendet. Zu Ihrer Erleichterung enthält die Originaldiskette 3 ein Beispielprogramm, DEMO2.BAS, das Sie verwenden können, um die folgende Übung zu vervollständigen.

Die folgenden Schritte zeigen, wie ein Programm in QuickBASIC geladen wird:

1. Betätigen Sie die ALT-TASTE.

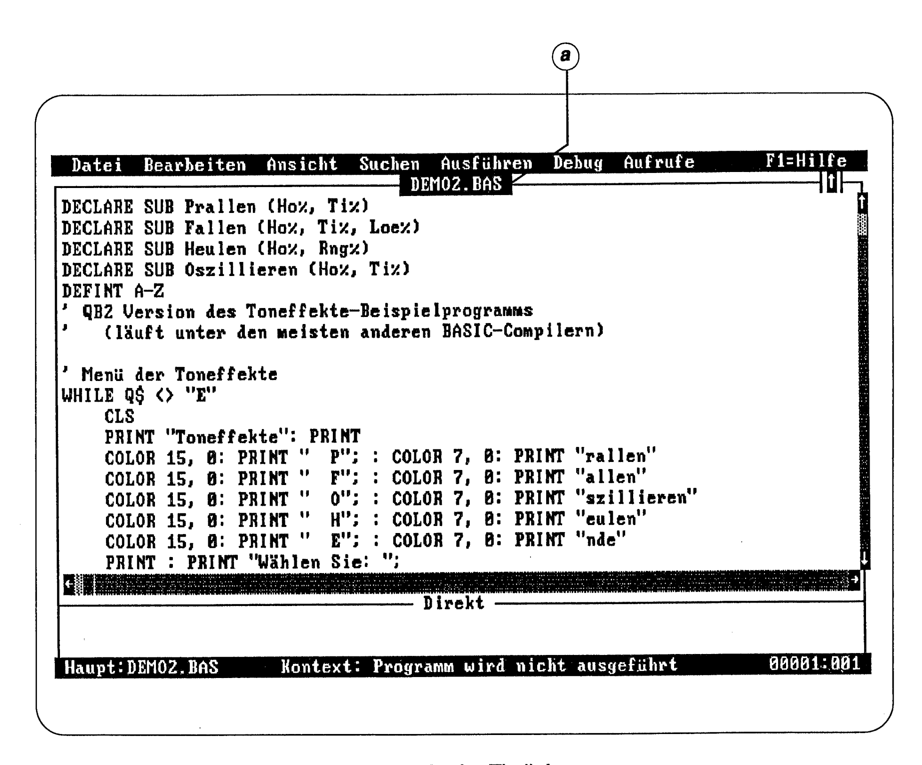
   Dies "markiert" (hebt hervor) den ersten Begriff (Datei) auf der Menüleiste. Beachten Sie, daß Sie die ALT-TASTE nicht gedrückt halten müssen. Wenn Sie mit Menüs arbeiten, können Sie die ALT-TASTE betätigen und loslassen, bevor Sie mit Schritt 2 fortfahren.

2. Betätigen Sie D, um das Menü **Datei** zu öffnen und betätigen Sie anschließend P, um das Dialogfeld des Befehls **Programm laden** anzuzeigen. Das Listenfeld in der Mitte zeigt alle BASIC-Programme, die im aktuellen Verzeichnis verfügbar sind.

3. Geben Sie den Namen des Programmes in das Textfeld mit einem vollständigen Pfadnamen ein, wenn das Programm, das Sie laden möchten, nicht aufgeführt ist. Andernfalls geben Sie nur seinen Namen ein und betätigen die EINGABETASTE.

   Wenn Ihr Programm mehr als einige wenige Zeilen hat, können Sie beobachten, wie QuickBASIC in dem Zähler in der unteren rechten Ecke des Bildschirms jede Zeile mitzählt, während das Programm geladen wird. Wenn Ihr Programm geladen ist, erscheint dessen Name zentriert in der "Titelleiste" des Fensters zwischen der Menüleiste und dem Beginn Ihres Codes (siehe Abbildung 2.5).

Abbildung 2.5 QuickBASIC-Bildschirm mit geladener DEMO2.BAS



a) Der Name des Programms erscheint in der Titelleiste.

2.2.3 Wo finden Sie Ihre Unterprogramme?

Diese QuickBASIC-Version enthält einen besonderen, "intelligenten" Editor. Dieser Editor erkennt die BASIC-Syntax und verwaltet untergeordnete Teile Ihres Programms, wie zum Beispiel **FUNCTION...END FUNCTION**- oder **SUB...END SUB**-Prozeduren, gesondert.

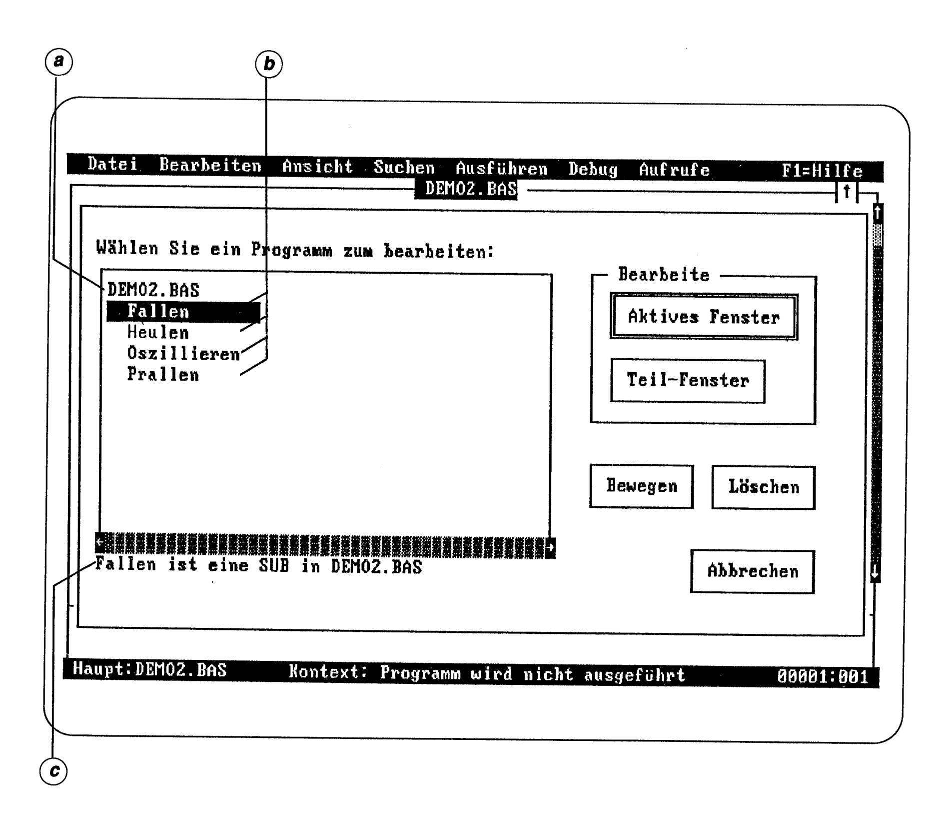
QuickBASIC faßt diese Prozeduren zusammen und behält sie in einem separaten Teil des Speichers. Sie sind dennoch Teil Ihres Programms und bereit, zusammen mit dem Modul-Ebenen-Code (das ist der Teil Ihres Programms, den Sie sehen, wenn Sie es zunächst laden) zu laufen.

Um sich Ihre **SUB**-Prozeduren anzusehen, gehen Sie wie folgt vor:

1. Betätigen Sie die ALT-TASTE und anschließend A, um das Menü **Ansicht** zu öffnen, und S, um den Befehl **SUBs** zu wählen (oder betätigen Sie F2, die Tastenkurzkombination).

   Es erscheint das in Abbildung 2.6 gezeigte Dialogfeld **SUBs**. Dieses Dialogfeld führt die aktuell im Speicher befindlichen Programmteile auf. Der Name des Programms erscheint oben in der Liste. Untergeordnete Prozeduren stehen eingerückt unter dem Programmnamen.

Abbildung 2.6 Das Dialogfeld ***SUBs***



a) Modulnamen erscheinen in Großbuchstaben.

b) Prozedurnamen stehen eingerückt unter dem Namen des Moduls, in dem sie erscheinen.

c) Beschreibt das markierte Modul bzw. die markierte Prozedur.

2. Betätigen Sie die NACH UNTEN (↓)-TASTE, um die Hervorhebung nach unten zu bewegen und eine **SUB** zu markieren; betätigen Sie die EINGABETASTE, um diese **SUB** zu betrachten oder zu bearbeiten.
3. Betätigen Sie die ALT-TASTE und anschließend A und S (oder betätigen Sie lediglich F2), um das Dialogfeld **SUBs** erneut anzuzeigen.
4. Der Programmname ist markiert, betätigen Sie daher die EINGABETASTE, um zum Modul-Ebenen-Code zurückzukehren.

Weitere Informationen zu den Merkmalen des Dialogfeldes **SUBs** finden Sie in Abschnitt 4.3.2, "Wie Sie mehrere Module anzeigen lassen".

2.2.4 Wie Sie ein BASIC-Programm starten

Wenn Sie eines Ihrer eigenen QuickBASIC-Programme erfolgreich geladen haben, sollten Sie in der Lage sein, es jetzt zu starten. Wenn Ihre Programme nicht korrekt geladen werden, verwenden Sie eines der Demonstrationsprogramme von der Originaldiskette.

Um das Programm mit dem Befehl **Start** aus dem Menü **Ausführen** zu starten

- betätigen Sie die ALT-TASTE, danach A und S (oder betätigen Sie lediglich UMSCHALTTASTE+F5).

Falls QuickBASIC eine Fehlermeldung erzeugt, wenn Sie Ihr Programm starten, versuchen Sie, ein anderes Programm zu laden und zu starten, so daß Sie die nächsten beiden Abschnitte vervollständigen können. Sobald Sie ein Programm vorliegen haben, das ohne Fehler startet, können Sie zum nächsten Abschnitt übergehen und Ihr Programm in eine ausführbare Datei übertragen, die direkt von der DOS-Eingabeaufforderung aus gestartet werden kann. (Informationen zur Veränderung alter Programme sowie die Unterschiede zwischen dieser und vorhergehenden QuickBASIC-Versionen finden Sie in Anhang A, "Übertragen von BASICA-Programmen nach QuickBASIC", sowie Anhang B, "Unterschiede zu früheren QuickBASIC-Versionen".)

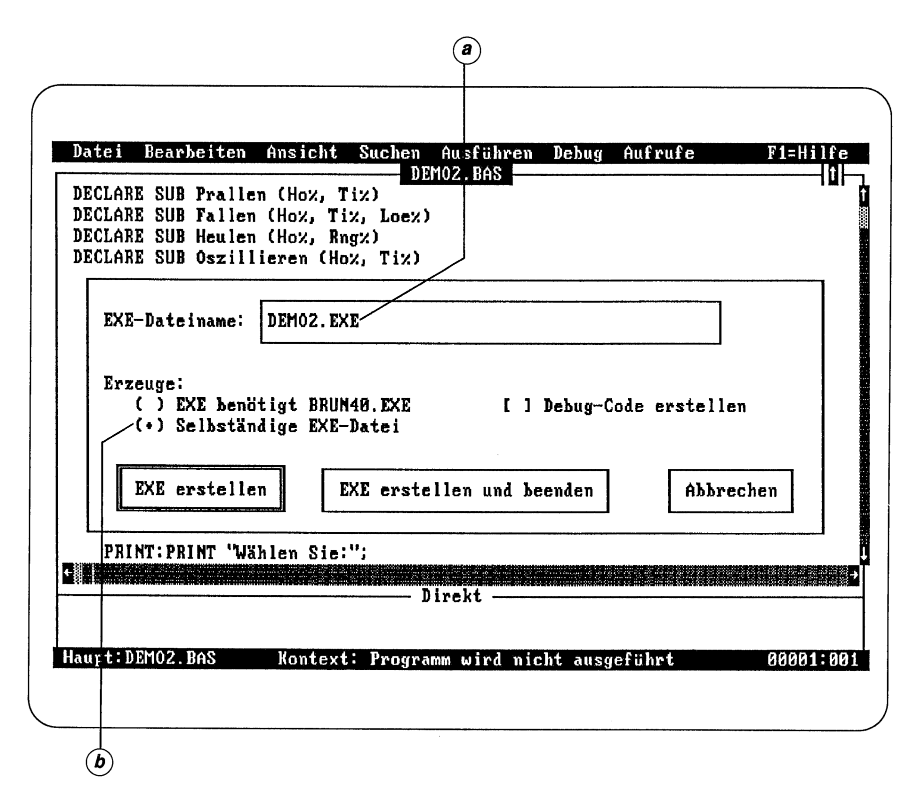
2.2.5 Wie Sie ein ausführbares Programm erstellen

Sie können Ihr Programm kompilieren und eine Datei erzeugen, die von der DOS-Eingabeaufforderung aus ausgeführt werden kann. Verwenden Sie den folgenden Ablauf, um aus QuickBASIC heraus ein ausführbares Programm auf Diskette zu erzeugen:

1. Betätigen Sie die ALT-TASTE und anschließend A, um das Menü **Ausführen** anzuzeigen.
2. Betätigen Sie E. (Wenn QuickBASIC Sie fragt, ob Sie die Datei speichern möchten, betätigen Sie J.) Es erscheint das Dialogfeld **EXE-Datei erstellen** (gezeigt in Abbildung 2.7). Der Name der ausführbaren Datei erscheint neben den Wörtern "EXE-Dateiname".



Abbildung 2.7 Das Dialogfeld ***EXE-Datei erstellen***



a) Geben Sie den Namen der zu erstellenden ausführbaren Datei ein.
b) Wählen Sie aus diesen beiden Optionen die Option "Selbständige EXE-Datei".

3. Halten Sie die ALT-TASTE gedrückt, während Sie S betätigen, um die Option "Selbständige EXE-Datei" zu wählen, sofern Sie den Dateinamen nicht ändern möchten.

4. Betätigen Sie U, um die Befehlsfläche "EXE erstellen und beenden" zu wählen.

QuickBASIC erstellt nun eine ausführbare Datei und bringt Sie zur DOS-Eingabeaufforderung zurück.

2.2.6 Wie Sie ein Programm aus DOS heraus ausführen

Verwenden Sie den DOS-Befehl DIR, um sich zu vergewissern, daß Ihre neue ausführbare Datei vorhanden ist. Sie sollten eine Datei finden, die den Namen hat, der in dem Textfeld "EXE-Dateiname" erschien, sowie die Erweiterung .EXE.

Um die selbständige Version Ihres Programmes zu starten, müssen Sie

1. hinter der DOS-Eingabeaufforderung deren Basisnamen eingeben,
2. die EINGABETASTE betätigen.

2.2.7 Verbesserungen in dieser QuickBASIC-Version

Obwohl ähnlich zu früheren Versionen – besonders im Erscheinungsbild und in der einfachen Verwendbarkeit – hat diese QuickBASIC-Version zusätzlich bedeutende Verbesserungen, besonders im Hinblick auf die beiden folgenden Kategorien:

1. Eine noch höher entwickelte "Programm-Entwicklungsumgebung", die Bearbeiten, Debuggen, Ausführen, Kompilieren und Bibliotheken erstellen integriert.
2. Bedeutende Erweiterungen der Sprache BASIC selbst.

Die Abschnitte 2.2.7.1 und 2.2.7.2 beschreiben diese kurz. Genauere Informationen zu diesen Verbesserungen finden Sie in Abschnitt 2.4.1, "Warum QuickBASIC besser ist als andere BASIC-Produkte".

2.2.7.1 Erweiterungen der Umgebung

Frühere QuickBASIC-Versionen trennten die Prozesse des Bearbeitens, Debuggens und Ausführens von Programmen. Mit dieser Version können alle diese Prozesse gleichzeitig stattfinden. Während Sie den Programmcode bearbeiten, übersetzen Sie gleichfalls jede Anweisung in ausführbaren Code und debuggen Ihr Programm. Dies schaltet den langwierigen Zyklus Bearbeiten-Kompilieren-Ausführen-Debuggen-Bearbeiten aus und führt zu einer schnelleren und leichteren Programmentwicklung.

- Der intelligente Editor überprüft die Syntax jeder Zeile, sobald Sie diese eingeben, und zeigt Ihnen den logischen Aufbau Ihres Programms, indem untergeordnete Prozeduren, wie zum Beispiel **SUB...END SUB**, in deren eigenen Fenstern für schnellen Zugriff und leichte Editierung angezeigt werden.
- Direkte Kompilierung bedeutet, daß jede Zeile beim Eingang im Speicher übersetzt wird, und Sie damit die meisten Syntaxfehler sofort feststellen und korrigieren können. Ihr Programm ist jederzeit bereit, gestartet zu werden.

- Mit interaktivem Debuggen können Sie ein laufendes Programm an jedem beliebigen Punkt anhalten, Variablenwerte überprüfen, das Programm bearbeiten und anschließend an dem Punkt fortfahren, an dem Sie das Programm angehalten haben. Wenn Sie Änderungen durchgeführt haben, die das Programm am Fortfahren hindern, können Sie wählen zwischen einem Fortfahren ohne Ihre Änderungen bzw. einem Start am Beginn des Programms mit den eingearbeiteten Änderungen.
- Ein Fenster für die direkte Ausführung, als Direkt-Fenster bezeichnet, ermöglicht es Ihnen, Programmanweisungen zu testen. Alles, was Sie im Direkt-Fenster eingeben, wird ausgeführt, sobald Sie die EINGABETASTE betätigen.
- QuickBASIC unterstützt jetzt mehrere Module im Speicher, so daß Sie Ihre gesamte Programmierung innerhalb der Umgebung durchführen können. Sobald Sie eine ausführbare Datei erstellen können, kompiliert und bindet QuickBASIC alle Module in einer einzigen Operation.
- Die automatische Bibliothekserstellung und -verwaltung mit Quick-Bibliotheken ermöglicht es Ihnen, die von Ihnen geschriebenen Prozeduren und Routinen in tatsächliche Erweiterungen der Sprache BASIC zu überführen, indem Sie lediglich einen einzigen Befehl aus dem Menü **Ausführen** wählen. Jedesmal, wenn Sie eine Quick-Bibliothek erstellen oder verändern, erzeugt oder aktualisiert QuickBASIC darüber hinaus automatisch eine entsprechende selbständige (.LIB) Bibliothek, die Sie von der DOS-Befehlszeile mit Programmen binden können.

2.2.7.2 Neue Sprachmerkmale

Die neuen Sprachmerkmale von QuickBASIC bieten größere Flexibilität und Leistungsfähigkeit für Ihre Programme:

- **FUNCTION**-Prozeduren bieten Ihnen eine weitere komfortable Möglichkeit, ein Programm in logisch unterschiedene Einheiten aufzubrechen, und erleichtern dadurch strukturierte Programmierung.
- Die Anweisung **TYPE...END TYPE** ermöglicht Ihnen, unterschiedliche Variablentypen zu einem zusammengesetzten, benutzerdefinierten Typ zusammenzufassen.
- Neue Konstruktionen zur Ablaufsteuerung helfen Ihnen, die Logik Ihres Programms übersichtlich und leicht veränderbar zu gestalten.
- Lange (32-Bit-) Ganzzahlen ermöglichen es Ihnen, Finanzberechnungen mit Pfennigen durchzuführen, so daß die Ergebnisse genau sind (keine Rundungsfehler). Diese Berechnungen können schneller durchgeführt werden, als es der Fall ist, wenn für die Berechnungen mit DM und Pfennigen Zahlen einfacher Genauigkeit verwendet werden.

- Rekursion bedeutet, daß sich **SUB**- und **FUNCTION**-Prozeduren selbst aufrufen können.
- Zahlen im IEEE-Format bieten zusätzlichen Umfang und zusätzliche Genauigkeit für Gleitkomma-Berechnungen.
- Binärer Zugriff auf Disketten-Dateien ermöglicht es Ihnen, Nicht-ASCII-Disketten-Dateien zu lesen und zu verändern.

2.2.8 Wie es weitergeht

Die Abschnitte 2.2.1 bis 2.2.7 haben Ihnen einen kurzen Überblick gegeben, wie sich diese QuickBASIC-Version von den QuickBASIC-Versionen 2.0 und 3.0 unterscheidet. Der nächste Schritt besteht darin, mehr über Menüs, Befehle, Dialogfelder und die Verwendung des Editors zu lernen. Die folgende Liste zeigt, wo Sie diese Informationen finden:

Information	*Fundstelle*
Weitere Informationen zu neuen Merkmalen	Abschnitt 2.4.1, "Warum QuickBASIC besser ist als andere BASIC-Produkte"
Beim Start von QuickBASIC zu verwendende Optionen	Abschnitt 2.4.2, "Der Befehl qb"
Die Verwendung von Menüs, Fenstern und Dialogfeldern	Kapitel 3, "Wie Sie sich in der QuickBASIC-Umgebung zurechtfinden"
Merkmale, die neu oder in dieser QuickBASIC-Version verbessert sind	Anhang B, "Unterschiede zu früheren QuickBASIC-Versionen"
Programmieren in BASIC mit neuen Sprachmerkmalen	*Programmieren in BASIC: Ausgewählte Themen*
Einrichten des QuickBASIC-Bildschirms	Abschnitt 2.4.3, "Einrichten des QuickBASIC-Bildschirms"
Das Hilfesystem von QuickBASIC benutzen	Abschnitt 2.4.5, "Wie Sie von QuickBASIC Hilfe erhalten"

2.3 Wenn Sie mit einem BASIC-Compiler gearbeitet haben

Beginnen Sie hier, wenn Sie Erfahrungen mit anderen BASIC-Compilern haben, oder wenn Sie zur Verarbeitung Ihrer Programme mit den QuickBASIC-Versionen 2.0 oder 3.0 die Methode der separaten Kompilierung verwenden.

In diesem Abschnitt werden Sie lernen, wie Sie ein Programm mit BC.EXE, dem selbständigen Befehlszeilen-Compiler von QuickBASIC, kompilieren und binden, und wo Sie Informationen zur Programmierumgebung und zu Spracherweiterungen von QuickBASIC finden können, um Programme zu schreiben, die leichter zu pflegen und zu debuggen sind.

2.3.1 Wie Sie von der Befehlszeile aus kompilieren und binden

Meistens werden Sie Ihre Programme wahrscheinlich innerhalb von QuickBASIC entwickeln wollen, dessen Vorteile im Abschnitt 2.4.1, "Warum QuickBASIC besser ist als andere BASIC-Produkte", behandelt werden. Sie können ein QuickBASIC-Programm jedoch auch von der DOS-Befehlszeile aus kompilieren und binden, wie im nächsten Abschnitt erläutert.

Bevor Sie weiterlesen, sollten Sie Ihre QuickBASIC-Software so installiert haben, wie es im Kapitel 1, "QuickBASIC installieren", beschrieben ist.

Für die folgende Übung können Sie eines Ihrer eigenen BASIC-Programme oder das mit QuickBASIC gelieferte Programm DEMO2.BAS verwenden. Wenn Sie für diese Übung Ihr eigenes Programm einsetzen, sollten Sie ein sehr einfaches Programm verwenden.

Wenn Sie während der Ausführung eines der folgenden Schritte zur Kompilierung Ihres Programms von der Befehlszeile aus eine Fehlermeldung erhalten, beginnen Sie erneut bei Schritt 1 mit dem Programm DEMO2.BAS, um diese Übung durchführen zu können:

1. Geben Sie den folgenden Befehl hinter der DOS-Eingabeaufforderung ein, um Ihr Programm zu kompilieren:

   `bc` *Dateiname*;

   Dieser Befehl erstellt eine Objektdatei mit dem Basisnamen der Quelldatei und der Erweiterung .OBJ, und anschließend erscheint wieder die DOS-Eingabeaufforderung.

Hinweis Es gibt viele Optionen, die Sie mit dem Befehl **bc** verwenden können (und einige *müssen* Sie verwenden; wenn das Programm, das Sie auswählen, zum Beispiel Fehler- oder Ereignisverfolgung einsetzt, müssen Sie dieses mit besonderen Optionen kompilieren). Weitere Informationen zu Optionen für **bc** finden Sie in Abschnitt 9.4.2.

2. Geben Sie die folgende Befehlszeile ein, um Ihr Programm zu binden:

   `link /e` *Dateiname*`.obj ;`

Dieser Schritt erzeugt eine ausführbare Datei mit dem Basisnamen der Quelldatei und der Erweiterung .EXE. Die Option **/e** "packt" die ausführbare Datei, so daß sie weniger Platz auf der Diskette erfordert und schneller geladen wird als eine "ungepackte" Datei. Eine umfassende Beschreibung aller mit dem Befehl **link** verfügbaren Optionen finden Sie in den Abschnitten 9.5.5 bis 9.5.6.

Versuchen Sie nun, das Programm zu starten. Wenn es mit früheren Microsoft BASIC-Versionen korrekt ausgeführt wurde, sollte es nun fehlerfrei laufen.

2.3.2 Wie es weitergeht

Dieser Abschnitt hat Ihnen eine Übersicht über den BC-Compiler sowie LINK gegeben. Nun sind Sie in der Lage, die QuickBASIC-Umgebung im Detail zu studieren und zu lernen, wie Sie diese für die Entwicklung Ihrer Programme einsetzen. Die folgende Liste zeigt Ihnen, wo Sie die benötigten Informationen finden:

Information	*Fundstelle*
Zusammenfassung neuer Umgebungs- und Sprachmerkmale	Abschnitt 2.4.1, "Warum QuickBASIC besser ist als andere BASIC-Produkte"
Beim Start von QuickBASIC zu verwendende Optionen	Abschnitt 2.4.2, "Der Befehl qb"
Die Verwendung von Menüs und Dialogfeldern	Kapitel 3, "Wie Sie sich in der QuickBASIC-Umgebung zurechtfinden"
Programmieren in BASIC mit neuen Sprachmerkmalen	*Programmieren in BASIC: Ausgewählte Themen*
Einrichten des QuickBASIC-Bildschirms	Abschnitt 2.4.3, "Einrichten des QuickBASIC-Bildschirms"
Das Hilfesystem von QuickBASIC benutzen	Abschnitt 2.4.5, "Wie Sie von QuickBASIC Hilfe erhalten"

2.4 Grundlegende Informationen für alle QuickBASIC-Benutzer

Der Rest dieses Kapitels enthält Informationen, die, unabhängig von Ihren BASIC-Kenntnissen, hilfreich für Sie sind. Die folgenden Abschnitte beschreiben die Verbesserungen an QuickBASIC und zeigen Ihnen, wie Sie QuickBASIC starten, das Aussehen Ihres QuickBASIC-Bildschirms einrichten und direkte (on-line) Hilfe von QuickBASIC erhalten.

2.4.1 Warum QuickBASIC besser ist als andere BASIC-Produkte

Die Verbesserungen der QuickBASIC-Programmier-Entwicklungsumgebung und seine Erweiterungen der Sprache BASIC ermöglichen es Ihnen, schnell Programme zu erstellen und zu debuggen, die mit maximaler Geschwindigkeit und Zuverlässigkeit laufen.

2.4.1.1 Die Entwicklungsumgebung, die Sie sich immer gewünscht haben

Bisher mußten BASIC-Programmierer wählen zwischen BASIC-Interpretern bzw. BASIC-Compilern. Der Vorteil eines BASIC-Interpreters, wie zum Beispiel BASICA, liegt darin, daß kein eigenständiger Schritt für die Übersetzung Ihres Codes in etwas für Ihren Computer Verständliches erforderlich ist. Daher ist der Übergang vom Schreiben und Debuggen zum Ausführen nahezu unverzüglich. Dennoch haben interpretierte Programme ihre Nachteile, die vorwiegend in ihrer langsamen Ausführungsgeschwindigkeit liegen.

Compiler überwinden diesen Geschwindigkeitsnachteil, allerdings auf Kosten des Programmierers. Weniger hochentwickelte BASIC-Compiler erfordern einen zeitraubenden Ablauf von Bearbeiten, Kompilieren, erneutem Bearbeiten, um Kompilierzeitfehler zu beseitigen, erneut kompilieren usw., bis Sie alle Kompilierzeitfehler beseitigt haben. Wenn dies geschehen ist, müssen Sie die kompilierte Datei noch mit Bibliotheken binden, die es der Datei erlauben, selbständig zu laufen, anschließend das Programm starten, um zu sehen, ob das Programm sich so verhält, wie Sie es erwarten. Wenn Laufzeitfehler auftreten, müssen Sie den gesamten Vorgang erneut durchführen.

QuickBASIC befreit Sie von dem Nachteil des Bearbeiten / Kompilieren / Binden / Debuggen-Ablaufs durch Integration dieser Schritte in eine nahtlose Verarbeitung, die allein im Speicher geschieht, anstatt mit physikalischen Dateien auf Diskette. Dies führt zu einer erheblich schnelleren und leichteren Programmentwicklung. QuickBASIC bietet den Komfort eines Interpreters, kombiniert mit der Leistungsfähigkeit und Ausführungsgeschwindigkeit eines Compilers.

Die folgende Liste beschreibt einige der Eigenschaften der QuickBASIC-Programmierumgebung, die es so einfach und bequem in seiner Verwendung machen:

- **Interaktiver Editor**

  Der Mittelpunkt der QuickBASIC-Umgebung ist ein intelligenter Editor, der die Syntax jeder Zeile überprüft, sobald Sie diese eingeben. Wenn die Syntax korrekt ist, wird die Zeile sofort in ausführbaren Code übersetzt, wenn nicht, erscheint eine Beschreibung des Fehlers. Darüber hinaus wandelt der Editor Schlüsselwörter in Großschreibung um und korrigiert bestimmte Auslassungen und Fehler. Zum Beispiel wird er am Ende einer **PRINT**-Anweisung ein Anführungszeichen einfügen, wenn es dort fehlt.

QuickBASIC zeigt außerdem den logischen Aufbau Ihres Programms. Zum Beispiel zeigt es untergeordnete Prozeduren wie **SUB...END SUB** in eigenen Fenstern für leichten Zugriff und schnelle Editierung an.

- **Direkte Ausführung**

  Da jede eingebene Zeile sofort in ausführbaren Code übersetzt wird, können Sie die meisten Fehler sofort entdecken und korrigieren. Darüber hinaus müssen Sie nicht auf das Kompilieren Ihres Programms warten, nachdem Sie das Bearbeiten Ihres Programmcodes beendet haben – es ist bereit, ausgeführt zu werden.

- **Interaktives Debuggen**

  Sie müssen nicht mehr zwischen Bearbeiten, Ausführen und Debuggen hin- und herschalten. Zum Beispiel können Sie ein laufendes Programm an jedem Punkt anhalten, das Programm bearbeiten, anschließend an dem Punkt fortfahren, an dem Sie das Programm angehalten haben. Wenn Sie Änderungen durchgeführt haben, die das Programm am Fortfahren an diesem Punkt hindern, können Sie wählen zwischen einem Fortfahren ohne Ihre Änderungen bzw. einem Start am Beginn des Programms mit den eingearbeiteten Änderungen.

  Das Direkt-Fenster am unteren Bildschirmrand ermöglicht es Ihnen, Programmanweisungen zu testen. Alles, was Sie in das Direkt-Fenster eingeben, wird ausgeführt, sobald Sie die EINGABETASTE betätigen. Sie können sogar temporäre Änderungen an Daten eines laufenden Programmes durchführen, so daß Sie Ihr Programm nicht erneut starten müssen, um die Auswirkungen einer Änderung zu sehen.

  Viele der Debug-Merkmale von QuickBASIC sehen genauso aus und arbeiten genauso wie diejenigen des Microsoft CodeView-Debuggers. Tatsächlich können Sie mit dem BC-Compiler Module erzeugen, die zu dem CodeView-Debugger kompatibel sind. Dadurch wird es einfach, die Module zu debuggen, wenn diese später in größere Projekte eingebunden werden.

- **Die schnelle Erstellung ausführbarer Dateien**

  Wenn Sie mit der Art, in der Ihr Programm innerhalb von QuickBASIC läuft, zufrieden sind, sind nur vier Tastenbetätigungen notwendig, um eine Version zu erstellen, die direkt von der DOS-Befehlszeile aus startet.

- **Mehrere Module im Speicher**

  Vor dieser QuickBASIC-Version konnten Sie mit mehreren Modulen nur programmieren, indem Sie die Module in QuickBASIC geschrieben haben und anschließend QuickBASIC verlassen haben, um das Programm mit einem BASIC-Befehlszeilen-Compiler sowie einem die Module zusammenbindenden Linker zu erstellen. QuickBASIC unterstützt nun mehr als ein Modul im Speicher. Wenn Sie sich entschließen, alle Module in eine ausführbare Datei zu kompilieren, führt QuickBASIC dies in einer einzigen Operation durch.

- **Automatische Erzeugung und Verwaltung von Bibliotheken**

  Quick-Bibliotheken ermöglichen es Ihnen, die von Ihnen geschriebenen Prozeduren und Routinen in tatsächliche Erweiterungen der BASIC-Sprache zu überführen. In dieser QuickBASIC Version erstellen Sie eine Quick-Bibliothek, indem Sie lediglich einen einzigen Befehl aus dem Menü **Ausführen** wählen. Jedesmal, wenn Sie eine Quick-Bibliothek erstellen oder verändern, erzeugt oder aktualisiert QuickBASIC darüber hinaus automatisch eine entsprechende selbständige (.LIB) Bibliothek, die Sie von der DOS-Befehlszeile aus mit Programmen binden können.

2.4.1.2 Neue Sprachmerkmale

Die neuen Sprachmerkmale von QuickBASIC bieten größere Flexibilität und Leistungsfähigkeit für Ihre Programme:

- **SUB- und FUNCTION-Prozeduren**

  Prozeduren unterstützen strukturierte Programmierung, indem Sie es Ihnen erlauben, ein Programm in logisch unterschiedene Einheiten aufzubrechen. Weitere Informationen finden Sie in Kapitel 2, "Prozeduren: Unterprogramme und Funktionen", in *Programmieren in BASIC: Ausgewählte Themen.*

- **TYPE...END TYPE**

  Die Anweisung **TYPE...END TYPE** erlaubt es Ihnen, unterschiedliche Variablentypen in einem zusammengesetzten, benutzerdefinierten Typ zusammenzufassen (dies ist ähnlich den "Strukturen" in der Sprache C und den "Verbunden" (Records) in Pascal). Die Anweisung **TYPE...END TYPE** erleichtert Eingaben von bzw. Ausgaben auf Direktzugriffsdateien sehr.

  Weitere Informationen zu benutzerdefinierten Datentypen finden Sie in Kapitel 2, "Datentypen", im *BASIC-Befehlsverzeichnis*; weitere Informationen darüber, wie benutzerdefinierte Typen Direktzugriffs-E/A vereinfachen, finden Sie in Kapitel 3, "Datei- und Geräte-E/A", in *Programmieren in BASIC: Ausgewählte Themen.*

- **Neue Konstruktionen zur Ablaufsteuerung**

  SELECT CASE, Block-**IF...THEN...ELSE** und **DO...LOOP** helfen Ihnen, die Logik Ihres Programms übersichtlich und leicht veränderbar zu halten.

  Weitere Informationen finden Sie in Kapitel 1, "Strukturen zur Ablaufsteuerung", in *Programmieren in BASIC: Ausgewählte Themen* sowie zu den Beschreibungen dieser Anweisungen im *BASIC-Befehlsverzeichnis.*

- **Lange (32-Bit-) Ganzzahlen**

  Lange Ganzzahlen ermöglichen Zahlen zwischen ungefähr plus/minus zwei Milliarden. Berechnungen, die mit ganzen Zahlen durchgeführt werden, die in diesem Bereich liegen, sind viel schneller und generell genauer als Gleitkomma-Berechnungen, die mit Zahlen einfacher Genauigkeit durchgeführt werden.

Lange Ganzzahlen ermöglichen es Ihnen, Finanzberechnungen mit Pfennigen durchzuführen, so daß die Ergebnisse genau sind (keine Rundungsfehler). Diese Berechnungen können schneller durchgeführt werden, als es der Fall ist, wenn für die Berechnungen mit DM und Pfennigen Zahlen einfacher Genauigkeit verwendet werden.

Weitere Informationen zu langen Ganzzahlen finden Sie in Kapitel 2, "Datentypen", im *BASIC-Befehlsverzeichnis.*

- **Rekursion**

  Rekursion bedeutet, daß **SUB**- und **FUNCTION**-Prozeduren sich selbst aufrufen können. Um mehr über diese Programmiertechnik zu erfahren, lesen Sie Kapitel 2, "Prozeduren: Unterprogramme und Funktionen", in *Programmieren in BASIC: Ausgewählte Themen* und Kapitel 4, "Programme und Module", im *BASIC-Befehlsverzeichnis.*

- **Zahlen im IEEE-Format**

  QuickBASIC verwendet nun Zahlen im IEEE-Format und bietet damit zusätzlichen Umfang und zusätzliche Genauigkeit für Gleitkomma-Berechnungen. Weitergehende Informationen finden Sie in Anhang B, "Unterschiede zu früheren QuickBASIC-Versionen".

- **Binärer Zugriff auf Disketten-Dateien**

  Binärer Zugriff ermöglicht es Ihnen, Nicht-ASCII-Disketten-Dateien zu lesen und zu verändern. Weitere Informationen finden Sie in Kapitel 3, "Datei- und Geräte-E/A", in *Programmieren in BASIC: Ausgewählte Themen.*

- **Befehlszeilenargumente testen**

  Frühere QuickBASIC-Versionen unterstützten die Funktion **COMMAND$**, die es erlaubt, Befehlszeilenargumente an ein kompiliertes Programm zu übergeben. Obwohl diese Funktion hilfreich ist für den Einsatz mit kompilierten Programmen, bestanden die Testmöglichkeiten für Programme, die **COMMAND$** verwenden, lediglich darin, das Programm entweder zu kompilieren oder QuickBASIC wiederholt mit unterschiedlichen Argumenten hinter der Option **/cmd** aufzurufen. (Weitere Informationen zu **/cmd** sowie anderen QuickBASIC-Optionen finden Sie in Abschnitt 2.4.2, "Der Befehl qb".) Wenn Sie jedoch den Befehl **Ändere COMMAND$** aus dem Menü **Ausführen** verwenden, können Sie nun **COMMAND$** innerhalb von QuickBASIC selbst verändern. Diese Eigenschaft ermöglicht es Ihnen, unterschiedliche Befehlszeilenargumente schnell und bequem zu testen.

2.4.2 Der Befehl qb

Der Befehl **qb** startet QuickBASIC. Darüber hinaus erlaubt es Ihnen dieser Befehl, eine Reihe von Optionen zu setzen, wie in der folgenden Syntax gezeigt:

qb [[**/run**] [*Quelldatei*] [**/b**] [**/g**] [**/h**] [**/c:***Puffergröße*] [**/l** [*Bibliotheksname*]] [**/mbf**] [**/ah**] [**/cmd** *Zeichenkette*]]

Die folgende Liste beschreibt die Argumente des Befehls **qb**:

Argument	***Beschreibung***
/run *Quelldatei*	Veranlaßt QuickBASIC, *Quelldatei* zu laden und auszuführen, bevor diese angezeigt wird.
Quelldatei	Benennt die beim Start von QuickBASIC zu ladende Datei. Weitere Informationen finden Sie in Abschnitt 2.4.6, "QuickBASIC-Vereinbarungen zur Dateibenennung".
/b	Erlaubt die Verwendung eines Schwarzweiß (Composite)-Monitors mit einer Farbgrafikkarte. Diese Option zeigt QuickBASIC auch dann in Schwarzweiß an, wenn Sie einen Farbmonitor einsetzen. Beachten Sie, daß Sie die Option **/b** nicht angeben müssen, wenn Sie eine monochrome Adapterkarte verwenden; QuickBASIC kann den Typ der Karte feststellen, nicht aber den Typ des Monitors.
/g	Modifiziert die Art, mit der QuickBASIC den Bildwechsel Ihres Bildschirms durchführt. Wenn Sie eine CGA einsetzen, paßt QuickBASIC automatisch die Art des Bildwechsels entsprechend Ihrer Hardware an. Auf einigen Maschinen ist es jedoch möglich, den Bildwechsel schneller durchzuführen als QuickBASIC dies tut. Die Option **/g** veranlaßt QuickBASIC dazu, den Bildwechsel so schnell wie möglich vorzunehmen. Falls Sie "Schnee" (flackernde Punkte auf dem Bildschirm) sehen, wenn Sie große Bereiche des Bildschirms aktualisieren (zum Beispiel mit BILD ↑), dann kann Ihre Hardware die hohe Geschwindigkeit nicht verarbeiten. Wenn Sie gegenüber dem schnellen Bildwechsel einen ruhigen Bildschirm bevorzugen, sollten Sie QuickBASIC ohne die Option **/g** starten. Diese Option betrifft nur Maschinen mit einer CGA.
/h	Zeigt die höchste Auflösung an, die mit Ihrer Hardware möglich ist. Wenn Sie zum Beispiel eine EGA haben, veranlaßt die Option **/h** QuickBASIC dazu, Text mit 43 Zeilen und 80 Spalten anzuzeigen.

Argument	*Beschreibung*
/c:*Puffergröße*	Setzt die Größe des Puffers, der Ferndaten empfängt. (Dem Übertragungspuffer sind 128 Bytes zugewiesen, und er kann nicht von der **qb**-Befehlszeile aus verändert werden.) Diese Option hat nur dann Auswirkungen, wenn eine asynchrone Datenübertragungskarte vorhanden ist. Die Standardpuffergröße ist 512 Bytes; die maximale Größe beträgt 32.767 Bytes. (Weitere Informationen zur asynchronen Datenübertragung finden Sie in Kapitel 3, "Datei- und Geräte-E/A", in *Programmieren in BASIC: Ausgewählte Themen*, sowie in der Beschreibung für **OPEN COM** im *BASIC-Befehlsverzeichnis.*)
/**l** *Bibliotheksname*	Lädt die Quick-Bibliothek, die mit *Bibliotheksname* angegeben ist. Wenn *Bibliotheksname* nicht angegeben ist, wird die Standard-Quick-Bibliothek, QB.QLB, geladen. Weitere Informationen finden Sie in Kapitel 8, "Quick-Bibliotheken".
/mbf	Veranlaßt die Konvertierungsfunktionen von QuickBASIC, Zahlen im IEEE-Format als Zahlen im Microsoft Binärformat zu behandeln. Die eingebauten Funktionen **MKS$**, **MKD$**, **CVS** und **CVD** arbeiten wie **MKSMBF$**, **MKDMBF$**, **CVSMBF** und **CVDMBF**. Weitere Informationen finden Sie in den Beschreibungen zu diesen Funktionen im *BASIC-Befehlsverzeichnis.*
/ah	Ermöglicht es dynamischen Verbunddatenfeldern, Zeichenketten fester Länge sowie numerischen Datenfeldern, jeweils größer als 64K zu sein. Falls diese Option nicht gewählt wurde, ist die maximale Größe für jedes Datenfeld 64K. Diese Option hat keine Auswirkung auf die Art, mit der Datengrößen an Prozeduren übergeben werden.
/cmd *Zeichenkette*	Übergibt *Zeichenkette* an die Funktion **COMMAND$**. Diese Option muß die letzte Option auf der Zeile sein. Weiterführende Informationen finden Sie in der Beschreibung der Funktion **COMMAND$** im *BASIC-Befehlsverzeichnis.*

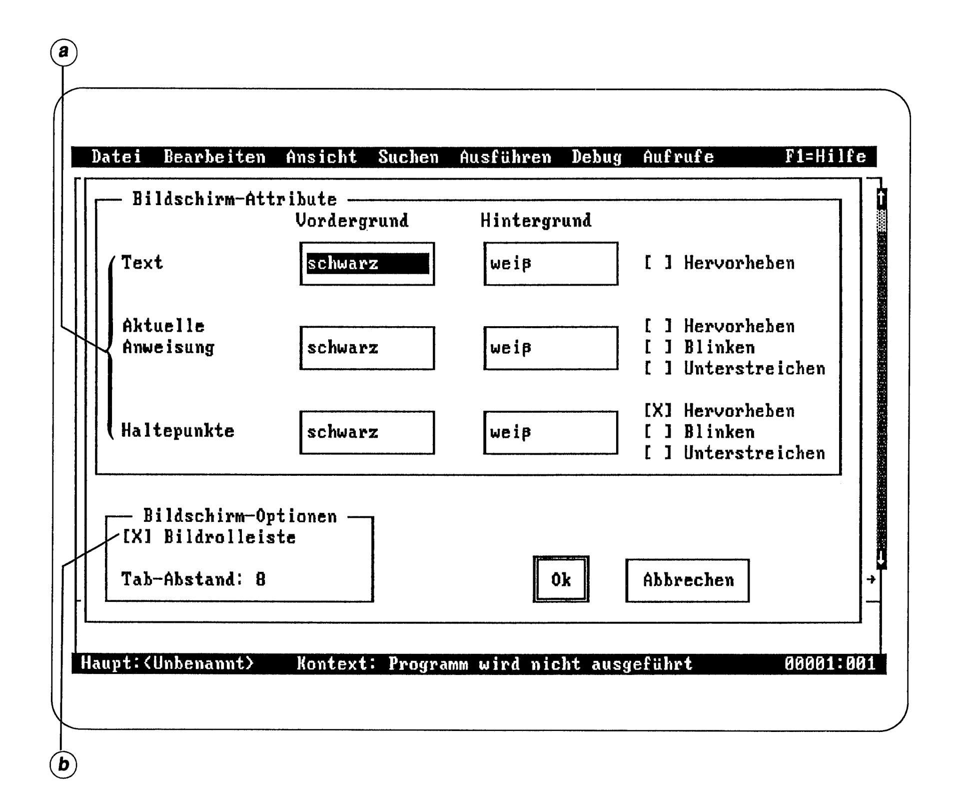
2.4.3 Einrichten des QuickBASIC-Bildschirms

Sie können den QuickBASIC-Bildschirm mit dem Befehl **Optionen** aus dem Menü **Ansicht** auf Ihren persönlichen Geschmack einrichten. Die Auswahl des Befehls **Optionen** läßt ein Dialogfeld erscheinen, mit dem Sie solche Dinge wie die Hintergrundfarbe und Vordergrundfarbe setzen können, und ob Sie es wünschen, daß Bildrolleisten unten oder an den Seiten des Bildschirms erscheinen, usw. Sie können sogar die Anzahl der Leerzeichen verändern, mit der die TAB-TASTE den Cursor bewegt.

Zum Beispiel zeigen Ihnen die folgenden Schritte, wie Sie die Hintergrundfarbe in QuickBASIC verändern:

1. Betätigen Sie die ALT-TASTE und anschließend N, um das Menü **Ansicht** zu öffnen.
2. Betätigen Sie O, um das Dialogfeld **Optionen** anzuzeigen, das in Abbildung 2.8 gezeigt ist.

Abbildung 2.8 Das Dialogfeld ***Optionen***



a) Verwenden Sie diese Optionen, um Ihre Bildschirmanzeige einzurichten.

b) Schalten Sie dieses Optionsfeld ein, wenn Sie Bildrolleisten auf dem Bildschirm sehen möchten.

3. Betätigen Sie die TAB-TASTE, um sich zwischen den Optionen und Feldern unter der Überschrift "Hintergrund" zu bewegen.
4. Verwenden Sie die NACH OBEN (↑) - und NACH UNTEN (↓) -RICHTUNGSTASTEN, um aus den Farben in den Feldern die Hintergrundfarbe auszuwählen.
5. Betätigen Sie die EINGABETASTE, wenn Sie alle gewünschten Optionen gesetzt haben.

Experimentieren Sie, um die Bildschirmkonfiguration zu erhalten, die Sie wünschen; Sie können mehr als eine Option auf einmal setzen. (Weitere Informationen, wie Optionen in Dialogfeldern markiert werden, finden Sie in Abschnitt 3.3.)

2.4.4 Die Datei QB.INI

Wenn Sie QuickBASIC verlassen, werden die Einstellungen, die Sie in dem Befehl **Optionen** aus dem Menü **Ansicht** gewählt haben, automatisch in einer QB.INI benannten Datei gespeichert. Solange Sie sich in dem Verzeichnis befinden, in dem QB.INI liegt, werden diese Einstellungen beim nächsten Start von QuickBASIC automatisch aufgerufen. (Alternativ dazu können Sie die Datei QB.INI in ein Verzeichnis bewegen, das in Ihrer PATH-Umgebungsvariablen aufgeführt ist. Auf diese Art und Weise erhalten Sie Ihre eingerichteten Einstellungen unabhängig von dem Verzeichnis, in dem Sie sich beim Start von QuickBASIC befinden. Um zu lernen, wie PATH-Umgebungsvariablen gesetzt werden, schlagen Sie in Abschnitt 1.7.1 nach.)

2.4.5 Wie Sie von QuickBASIC Hilfe erhalten

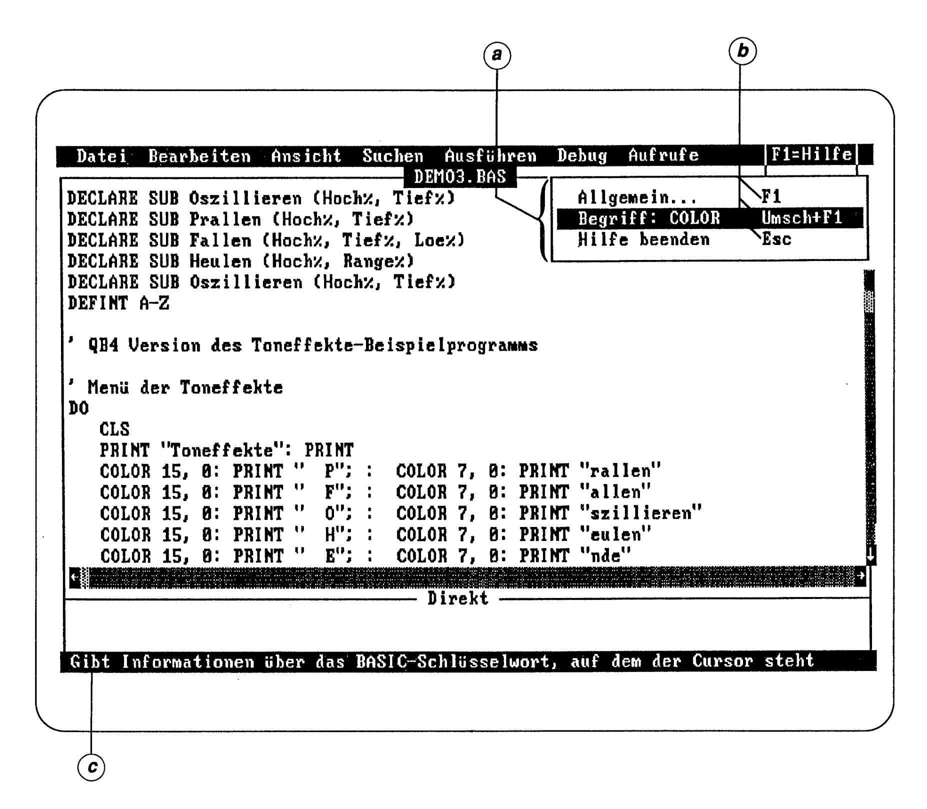
QuickBASIC kann Ihnen bei Fragen zu Tastenkurzkombinationen helfen, die Sie für Editier- oder Debug-Aufgaben benötigen, oder Ihnen helfen, die Syntax einer Anweisung oder Funktion zu überprüfen.

QuickBASIC kennt drei Arten von direkter (on-line) Hilfe:

1. Hilfe zu QuickBASIC-Menübefehlen

   Diese Hilfe besteht aus einer kurzen Beschreibung einer aktuell hervorgehobenen Wahl eines geöffneten Menüs (siehe Abbildung 2.9). Um diese Beschreibung zu erhalten, ist es nicht notwendig, eine Taste zu betätigen oder eine Menüwahl zu treffen; die Beschreibung erscheint stets auf der Statuszeile im unteren Teil Ihres Bildschirms, sobald ein Menü geöffnet ist.

Abbildung 2.9 Hilfe zu Menübefehlen

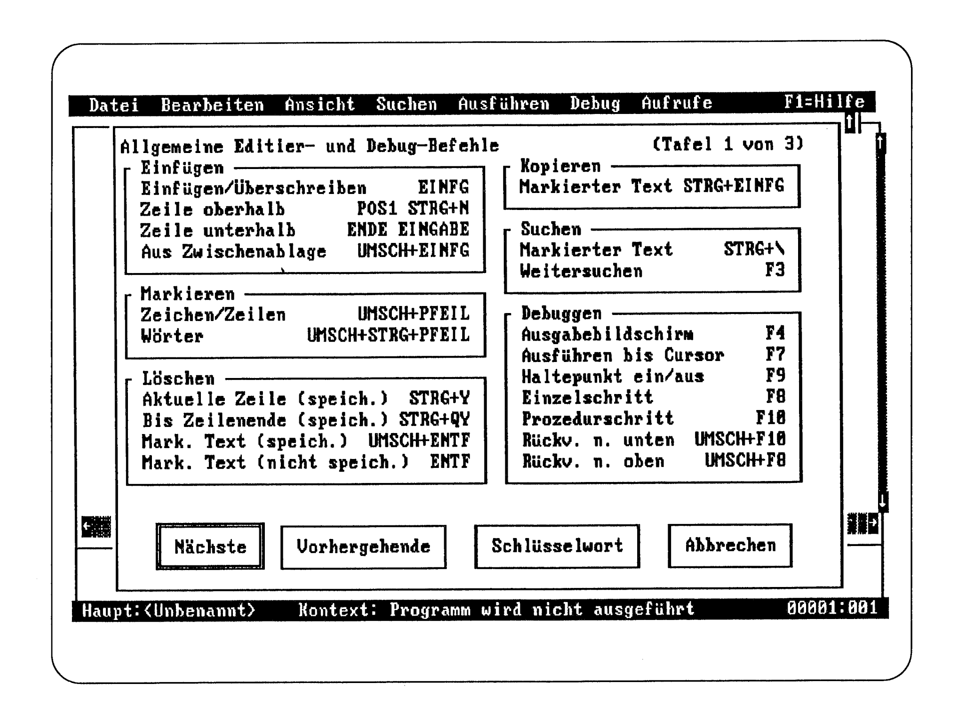


*a) Das Menü **Hilfe***
b) Tastenkurzkombinationen
c) Hilfeinformation zu dem hervorgehobenen Menübefehl

2. Allgemeine Hilfe.

Allgemeine Hilfe ist durch das ganz rechts auf der Menüleiste liegende Menü **Hilfe** (betätigen Sie ALT+H) verfügbar (siehe Abbildung 2.9 weiter oben). Allgemeine Hilfe beschreibt Dinge wie das Bearbeiten von Prozeduren und Tastenkurzkombinationen (siehe Abbildung 2.10). (Sie können jedoch auch die Taste F1 betätigen, um allgemeine Hilfe zu erhalten.)

Abbildung 2.10 Allgemeine Hilfe



3. Kontext-sensitive Hilfe.

Kontext-sensitive Hilfe bietet Ihnen Informationen zur BASIC-Syntax. Um diese Hilfe zu erhalten, positionieren Sie den Cursor auf dem Schlüsselwort (oder auf dem Leerschritt direkt hinter dem Schlüsselwort), über das Sie Informationen wünschen, und betätigen Sie anschließend UMSCHALTTASTE+F1, um eine Übersicht über die Syntax der Anweisung zu erhalten.

Kontext-sensitive Hilfe kann ebenso durch den Befehl **Begriff** aus dem Menü **Hilfe** abgerufen werden.

2.4.6 QuickBASIC-Vereinbarungen zur Dateibenennung

QuickBASIC nimmt für Dateien, die Sie angeben, bestimmte Voraussetzungen an. Dieser Abschnitt beschreibt diese Voraussetzungen sowie andere Regeln, die Sie bei der Benennung und bei der Angabe von Dateien beachten sollten.

2.4.6.1 Pfadnamen

Jeder Dateiname kann einen vollständigen oder teilweisen Pfadnamen enthalten. Ein vollständiger Pfadname beginnt mit der Laufwerksangabe; ein Teilpfadname besteht aus einem oder mehreren Verzeichnisnamen vor dem Dateinamen, enthält jedoch keine Laufwerksangabe. Ein vollständiger Pfadname erlaubt es Ihnen, Dateien auf unterschiedlichen Laufwerken und in unterschiedlichen Verzeichnissen anzugeben.

Beispiele

```
a:prog\grafik\kreis.bas      ' ein vollständiger Pfadname
\grafik\kreis.bas            ' ein Teilpfadname
..\grafik\kreis.bas          ' ein Teilpfadname
```

2.4.6.2 Groß- und Kleinbuchstaben

Innerhalb von QuickBASIC können Sie beliebige Kombinationen von Groß- und Kleinbuchstaben für Dateinamen verwenden, wie im folgenden Beispiel gezeigt, in dem sich alle Namen auf dieselbe Datei beziehen:

Beispiele

```
abcde.bas
ABCDE.BAS
aBcDe.Bas
```

2.4.6.3 Erweiterungen von Dateinamen

Ein DOS-Dateiname besteht aus zwei Teilen: Dem "Basisnamen", der aus allem bis zu dem Punkt (.) besteht, ohne diesen zu enthalten, sowie der "Erweiterung", die den Punkt und bis zu drei Zeichen, die dem Punkt folgen, umfaßt. Im allgemeinen kennzeichnet die Erweiterung den Typ der Datei, wie zum Beispiel eine BASIC-Quelldatei oder eine ausführbare Datei.

Die Standard-Dateierweiterungen lauten wie folgt:

Erweiterung	*Erläuterung*
.BAS	Eine BASIC-Quelldatei
.EXE	Eine ausführbare Datei (weitere Informationen finden Sie in Kapitel 6, "Wie Sie QuickBASIC-Programme erstellen und ausführen" und in Kapitel 9, "Wie Sie aus DOS heraus kompilieren und binden")
.QLB	Eine Quick-Bibliotheksdatei (weitere Informationen finden Sie in Kapitel 8, "Quick-Bibliotheken")
.LIB	Eine selbständige Bibliotheksdatei (weitere Informationen finden Sie in Kapitel 8, "Quick-Bibliotheken" und in Kapitel 9, "Wie Sie aus DOS heraus kompilieren und binden")

Beispiele

```
kreis.bas     ' eine BASIC-Quelldatei
kreis.exe     ' eine ausführbare Programmdatei
kreis.qlb     ' Quick-Bibliothek
kreis.lib     ' eine selbständige Bibliothek
```

2.4.6.4 Wie Sie Dateinamen angeben

QuickBASIC nimmt für Dateinamen die folgenden Voraussetzungen an:

- Falls keine Erweiterung angegeben ist, nimmt QuickBASIC die Erweiterung .BAS an, wenn Sie den Befehl **qb** mit dem Argument *Quelldatei* verwenden, und wenn Sie neue Programme starten und Dateien öffnen.
- Wenn Sie eine ausführbare Datei erstellen, nimmt QuickBASIC an, daß diese die Erweiterung .EXE haben wird.
- Wenn Sie die Option **/l** (Quick-Bibliothek laden) verwenden, nimmt QuickBASIC die Erweiterung .QLB an.

Hinweis Wenn Sie eine Datei angeben möchten, die keine Erweiterung hat, schreiben Sie an das Ende des Dateinamens einen Punkt. Zum Beispiel, wenn Sie QuickBASIC starten und eine PROG benannte Datei laden möchten, geben Sie folgendes ein, um QuickBASIC daran zu hindern, nach PROG.BAS zu suchen:

```
PROG.
```

Teil 2: Die QuickBASIC-Programm-Entwicklungsumgebung

Dieser Teil des Handbuchs enthält detaillierte Anweisungen zur Verwendung der QuickBASIC-Programm-Entwicklungsumgebung, die deren Bildschirme, Menüs sowie weiteren Eigenschaften umfassen. Er erläutert, wie QuickBASIC-Programmdateien und -Module geladen, gespeichert und verwaltet werden; außerdem wird erläutert, wie die fortgeschrittenen Eigenschaften des Editors benutzt und wie ausführbare Dateien und mehrmodulige Programme erstellt werden. Dieser Teil beschreibt auch, wie Sie Ihre Programme untersuchen und debuggen, selbst während diese laufen.

3 Wie Sie sich in der QuickBASIC-Umgebung zurechtfinden

Dieses Kapitel stellt die QuickBASIC-Programm-Entwicklungsumgebung vor und veranschaulicht viele ihrer Eigenschaften. Sie müssen jedoch nicht alle QuickBASIC-Eigenschaften verstehen, bevor Sie beginnen, Ihre Programme mit der QuickBASIC-Umgebung zu schreiben. Dieses Kapitel vermittelt Ihnen die nötigen Grundlagen, damit Sie beginnen können.

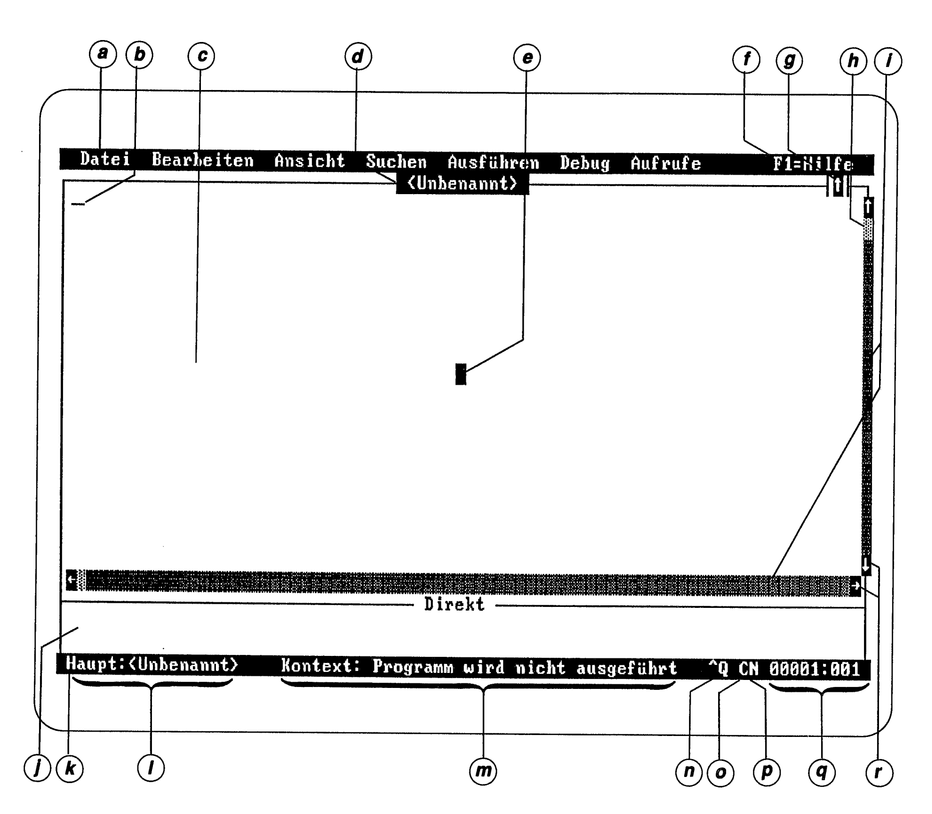
In diesem Kapitel werden Sie lernen, wie

- Befehle aus Menüs gewählt werden
- Objekte aus Dialogfeldern gewählt werden
- Listings gerollt werden
- Text in einem Dialogfeld oder Fenster markiert wird
- die Fenstergröße verändert wird
- eine oder mehrere BASIC-Anweisungen eingegeben und direkt ausgeführt werden

3.1 Der QuickBASIC-Bildschirm

Wenn Sie QuickBASIC ohne ein Dateinamenargument starten, erscheint Ihr Bildschirm wie in Abbildung 3.1 gezeigt.

Abbildung 3.1 Der QuickBASIC-Bildschirm



Die folgende Liste erläutert die Teile des Bildschirms und bezieht sich auf Abbildung 3.1:

a) Menüleiste. Benennt jedes Menü. Hervorgehobene Buchstaben geben die Tastenkurzkombination an.

b) Der Cursor. Blinkendes Unterstreichungszeichen, das anzeigt, wo der Text, den Sie eingeben, erscheinen wird.

c) Arbeitsbereich. Ein Fenster in Ihrem Programmtext. (Kann in zwei Fenster geteilt werden.)

d) Titelleiste. Benennt Dateien (und Prozeduren, sofern vorhanden), die aktuell im Fenster darunter gezeigt werden.

e) Mauszeiger. Zeigt die aktuelle Bildschirmposition der Maus (nur Maus).

f) F1 = Hilfe. Zeigt die zu betätigende Taste an, um direkte Hilfe zu erhalten.

g) Vergrößerungsfeld. Erweitert das aktive Fenster, so daß es den gesamten Bildschirm einnimmt (nur zur Verwendung mit der Maus).

h) *Bildrollfeld. Zeigt die relative Cursorposition innerhalb der Datei oder Prozedur.*
i) *Bildrolleisten. Rollen von Text im aktiven Fenster (nur Maus).*
j) *Direkt-Fenster. Bereich, in dem BASIC-Anweisungen direkt ausgeführt werden können.*
k) *Statusleiste. Enthält Textposition, Tastaturstatus und Informationen zu dem Programm.*
l) *Haupt. Zeigt den Namen des Hauptmoduls Ihres Programmes an.*
m) *Kontext. Benennt das Modul (und die Prozedur, sofern vorhanden), das die Anweisung enthält, die als nächste ausgeführt wird.*
n) *^Q. Erscheint, wenn diese WordStar entsprechende Befehlssequenz eingegeben wird. ^K erscheint hier, wenn Sie eine Textmarkierung setzen, und ^P erscheint hier, wenn Sie ein Kontrollzeichenliteral eingeben. (Weitere Informationen finden Sie in Kapitel 5, "Bearbeiten".)*
o) *C. Erscheint, wenn die UMSCHALT-FESTSTELLTASTE eingeschaltet ist.*
p) *N. Erscheint, wenn die NUM-FESTSTELLTASTE eingeschaltet ist.*
q) *Zeilen- und Spaltenzähler. Geben die aktuelle Position des Cursors innerhalb des Textes an, der sich im aktiven Fenster befindet.*
r) *Rollpfeile. Rollen den Text Zeichen für Zeichen oder Zeile für Zeile (nur Maus).*

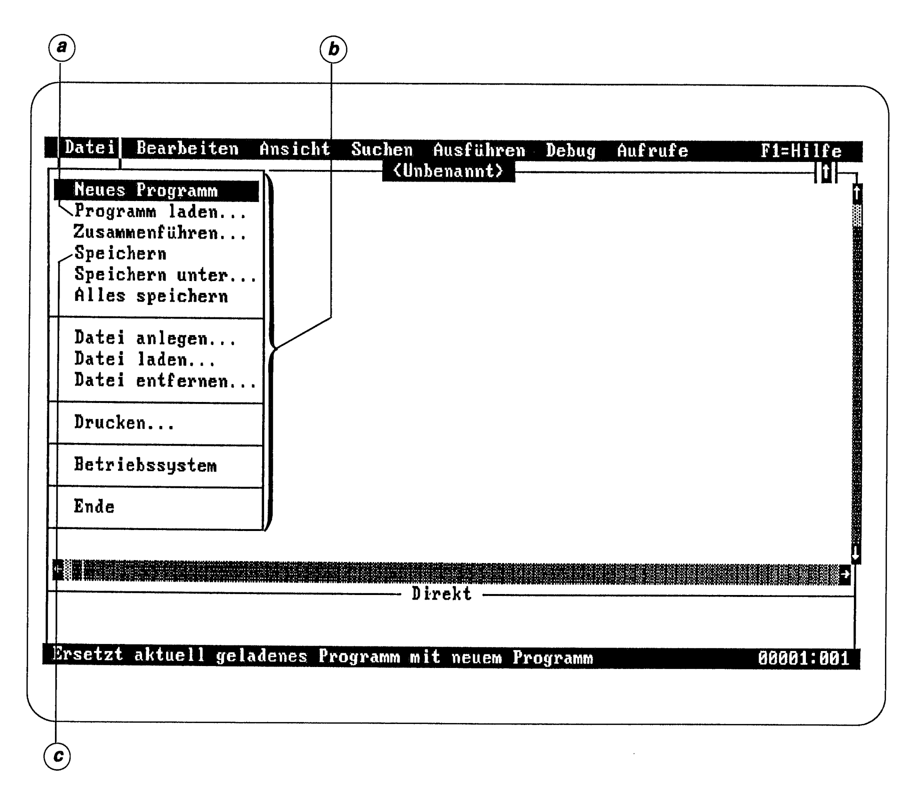
Nun können Sie beginnen, eigene Programme zu schreiben, oder Sie können von QuickBASIC weitere Hilfen erhalten. Um das Direkt-Hilfesystem von QuickBASIC zu benutzen, betätigen Sie F1, um den ersten aus einer Reihe von Hilfebildschirmen aufzurufen, die die Tastenzuweisungen und BASIC-Syntax erklären.

Um den Hilfebildschirm zu löschen, betätigen Sie die ESC-TASTE. Das Betätigen von ESC löscht ebenfalls Menüs, Dialogfelder und Fehlermeldungen von dem QuickBASIC-Bildschirm. (Weitere Informationen zur direkten Hilfe von QuickBASIC finden Sie in Abschnitt 2.4.5.)

3.2 Wie Sie Menüs öffnen und Befehle wählen

QuickBASIC-Befehle sind in Menüs auf der Menüleiste angeordnet. Abbildung 3.2 zeigt eines dieser Menüs, das Menü **Datei**.

*Abbildung 3.2 Das Menü **Datei***



a) Befehle, denen Punkte folgen, zeigen Dialogfelder an, in die Sie zusätzliche Informationen eingeben können.

*b) Menü **Datei***

c) Befehle ohne Punkte werden sofort ausgeführt.

Die QuickBASIC-Umgebung ist für schnelle, einfache Operationen entwickelt. Die meisten Operationen in QuickBASIC können Sie mit jeder der folgenden Techniken durchführen, wählen Sie daher diejenige, die Ihnen am bequemsten erscheint:

- Menüs öffnen und Befehle wählen, entweder mit der Tastatur oder mit der Maus, wenn Sie eine solche besitzen.
- Befehle direkt ausführen mit Tastenkurzkombinationen und normalen Tastenkombinationen.

3.2.1 Tastaturtechnik

Wenn Sie die Tastatur verwenden, können Sie jeden Befehl eines QuickBASIC-Menüs wählen, indem Sie die folgenden Schritte durchführen:

1. Öffnen des Menüs.

   Betätigen Sie die ALT-TASTE, lassen Sie diese wieder los, und betätigen Sie anschließend den ersten Buchstaben des Menünamens. Dieses "markiert" (hebt hervor) den gesamten Namen.

Hinweis Wenn Sie mit Menüs arbeiten, müssen Sie die ALT-TASTE nicht gedrückt halten. Die ALT-TASTE ist ein "Umschalter", so daß durch erneute Betätigen der ALT-TASTE die Hervorhebung verschwindet. Diese Eigenschaft der ALT-TASTE, "betätigen und loslassen", ist nur wirksam im Zusammenhang mit Menüs. Wenn Sie Tastenkombinationen in anderen Zusammenhängen verwenden, müssen Sie die erste Taste gedrückt halten, während Sie den Rest der Sequenz eingeben.

   Wenn Sie das falsche Menü öffnen, führen Sie einen der folgenden Schritte durch:

   - Schließen Sie das Menü durch Betätigen von ESC, und versuchen Sie anschließend Schritt 1 erneut.
   - Gehen Sie zu einem anderen Menü, indem Sie die NACH LINKS (←)- oder die NACH RECHTS (→)-Richtungsstaste betätigen.

2. Wählen des Befehls.

   Menünamen und Befehlsnamen enthalten immer einen hervorgehobenen Buchstaben. Auf Farbbildschirmen hat dieser Buchstabe eine von den anderen Buchstaben des Namens verschiedene Farbe; auf monochromen Monitoren ist dieser Buchstabe unterstrichen, wie in Abbildung 3.2 gezeigt. Führen Sie einen der folgenden Schritte durch, um zu dem gewünschten Befehl zu gelangen:

   - Betätigen Sie die Taste, die dem hervorgehobenen (oder unterstrichenen) Buchstaben entspricht.
   - Verwenden Sie die NACH OBEN (↑)- oder die NACH UNTEN (↓)-Richtungstasten, um die Hervorhebung zu dem Befehl zu bewegen, den Sie wünschen, und betätigen Sie anschließend die EINGABETASTE.

Wenn in einem Befehl kein Buchstabe hervorgehoben ist, können Sie diesen Befehl solange nicht wählen, bis Sie einige andere Schritte durchgeführt haben. Zum Beispiel, wenn Sie einen Programmtext bearbeiten, aber keinen Text zum Kopieren verwenden oder Löschen markiert haben, dann können Befehle wie **Ausschneiden** oder **Kopieren** in dem Menü **Bearbeiten** keine Auswirkung haben und erscheinen daher ohne einen hervorgehobenen Buchstaben. Sobald Sie Text markiert haben, können Sie die Befehle **Ausschneiden** und **Kopieren** verwenden. (Schlagen Sie in Kapitel 5, "Bearbeiten", nach, um zu lernen, wie Text zum Kopieren oder Löschen markiert wird.)

Einige Befehle, wie zum Beispiel der Befehl **Neues Programm** aus dem Menü **Datei**, wirken sich direkt aus und können, nachdem sie einmal gewählt sind, nicht abgebrochen werden. Andere dagegen veranlassen QuickBASIC dazu, ein Dialogfeld zu öffnen, in dem Sie zusätzliche Informationen zur Verfügung stellen können. Betätigen Sie ESC, um einen solchen Befehl abzubrechen. Erläuterungen zu Dialogfeldern finden Sie in Abschnitt 3.3.

3.2.2 Maustechnik

Um Befehle mit der Maus zu wählen

1. Zeigen Sie auf den Menünamen, und klicken Sie anschließend auf die linke Maustaste.
2. Zeigen Sie auf den Befehl, und klicken Sie anschließend auf die linke Maustaste.

Um einen Befehl mit der Maus abzubrechen

- Bewegen Sie den Zeiger von dem Menü weg, und klicken Sie die linke Maustaste.

Hinweis Verwenden Sie nur die linke Maustaste, um QuickBASIC-Menüs oder -Befehle zu wählen. Verwenden Sie die rechte Maustaste, wenn Sie Ihre Programme debuggen; nach der Betätigung der rechten Maustaste wird Ihr Programm immer bis zu der Zeile ausgeführt, auf der sich der Mauszeiger befindet.

Die folgende Tabelle erläutert Maustechniken, mit denen andere in diesem Kapitel erläuterte Bildschirmaufgaben erledigt werden.

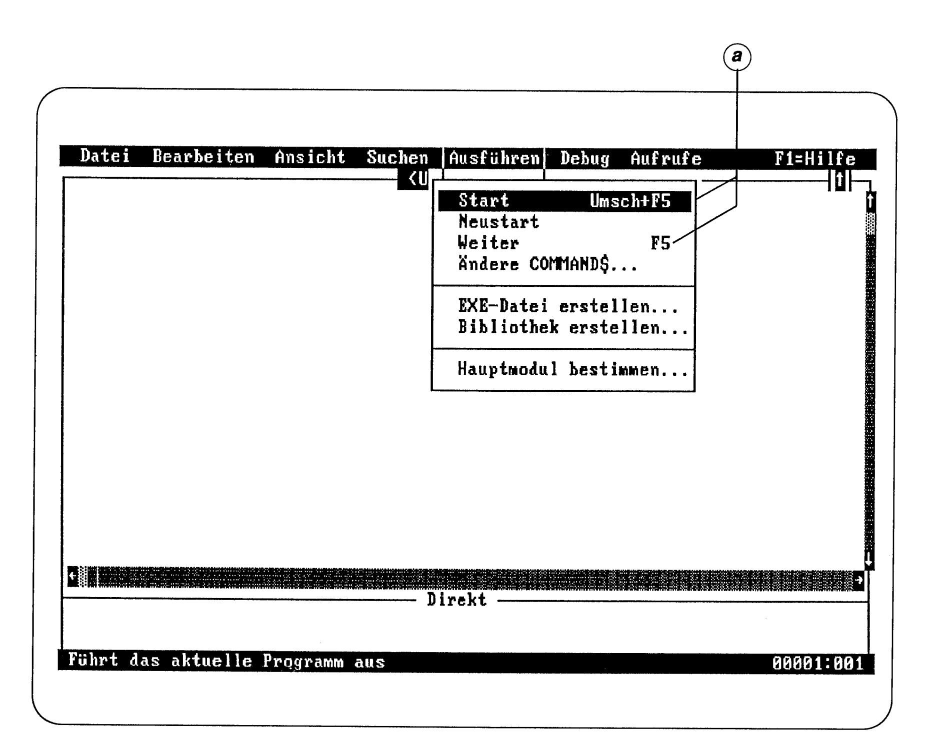
Aufgabe	*Maustechnik*
Ein Fenster aktiv werden lassen (siehe Abschnitt 3.4.3)	Klicken Sie irgendwo in dem Fenster.
Ein aktives Fenster vergrößern oder verkleinern (siehe Abschnitt 3.4.4)	Bewegen Sie den Mauszeiger auf die Titelleiste, und ziehen Sie die Titelleiste herauf oder hinunter.
Das aktive Fenster so vergrößern, daß es den gesamten Bildschirm einnimmt (siehe Abschnitt 3.4.4)	Klicken Sie das Vergrößerungsfeld (|↑|) rechts auf der Titelleiste, oder doppelklicken Sie die Maus irgendwo auf der Titelleiste.
Text rollen (siehe Abschnitt 3.4.5)	Zeigen Sie auf das Rollfeld, und ziehen Sie dieses auf eine Position auf der Leiste, die ungefähr der von Ihnen gewünschten Stelle in der Datei entspricht. Um Seite für Seite zu rollen, plazieren Sie den Mauszeiger auf der Rolleiste zwischen dem Rollfeld und dem oberen oder unteren Teil der Rolleiste und klicken die linke Taste. Um Zeile für Zeile oder Zeichen für Zeichen zu rollen, klicken Sie die Rollpfeile an einem der beiden Enden der Rolleisten.

Weitere Informationen zu Mausbefehlen finden Sie in Anhang D, "Zusammenfassung der Befehle".

3.2.3 Wie Sie Tastenkurzkombinationen verwenden

Die Tastenkurzkombinationen für einen Menübefehl werden immer in dem Menü neben dem Befehl aufgeführt. Während Sie mit QuickBASIC vertraut werden, werden Sie viele solcher Tastenkurzfolgen lernen. Um ein Programm zu starten, können Sie zum Beispiel den Befehl **Start** aus dem Menü **Ausführen** wählen, oder Sie können das Menü umgehen und die Tastenkurzkombination UMSCHALTTASTE+F5 verwenden (siehe Abbildung 3.3). Eine Liste der Tastenkurzkombinationen von QuickBASIC finden Sie in Anhang D, "Zusammenfassung der Befehle", sowie in Abschnitt D.2, "Menübefehle und Tastenkurzkombinationen".

Abbildung 3.3 Tastenkurzkombinationen



a) Tastenkurzkombinationen

3.3 Wie Sie Dialogfelder benutzen

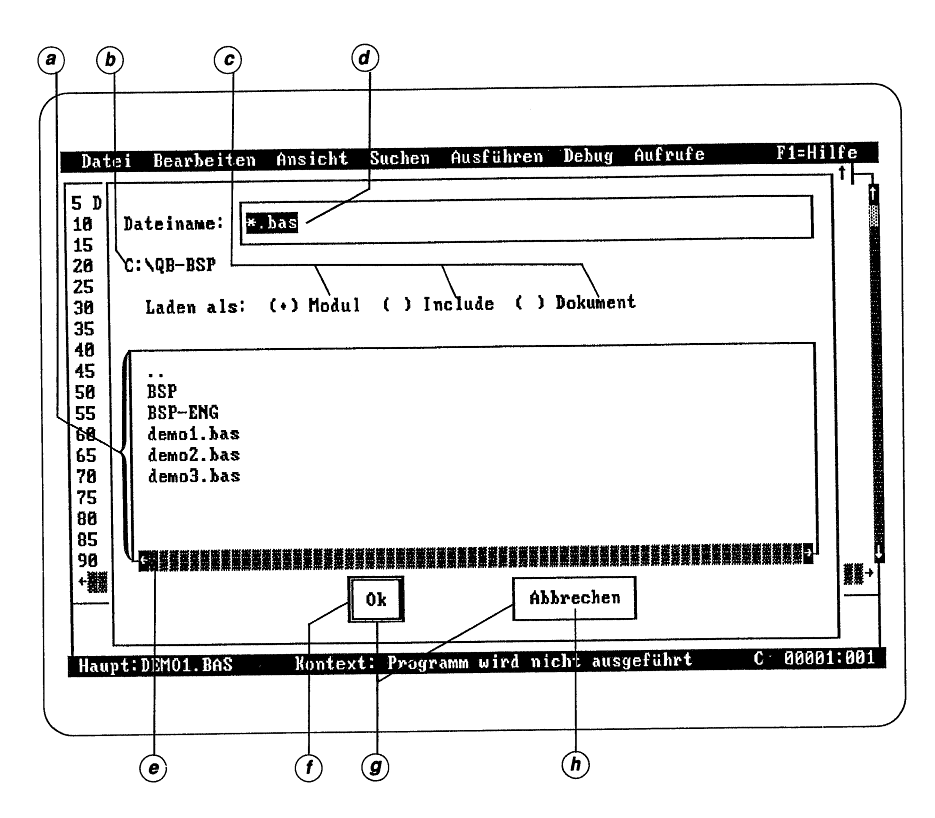
QuickBASIC zeigt ein Dialogfeld, wenn es zusätzliche Informationen von Ihnen benötigt, um eine Aufgabe ausführen zu können. Zum Beispiel kann ein Dialogfeld

- Sie nach einem Dateinamen fragen
- Ihnen eine Liste von wählbaren Optionen anzeigen
- Sie auffordern, einen Befehl zu bestätigen oder abzubrechen

In diesem Abschnitt verdeutlichen die in den Abbildungen 3.4 und 3.5 gezeigten Dialogfelder **Datei laden** aus dem Menü **Datei** und **Optionen** aus dem Menü **Ansicht** die Teile eines Dialogfeldes.



Abbildung 3.4 Das Dialogfeld ***Datei laden***



a) Listenfeld. Betätigen Sie die TAB-TASTE, um sich in dieses Feld zu bewegen, die RICHTUNGSTASTEN oder den ersten Buchstaben, um sich zu einem Datei- oder Unterverzeichnisnamen zu bewegen, betätigen Sie die EINGABETASTE, um die Wahl auszuführen.

b) Verzeichnisname der angezeigten Dateien.

c) Optionsfläche. Es ist wichtig, beim Laden von Dateien die richtige Option zu setzen. Betätigen Sie TAB, um sich auf diese Zeile zu bewegen, RICHTUNGSTASTEN, um die Option zu verändern, TAB, um die Option zu markieren.

d) Textfeld. Geben Sie hier Informationen ein.

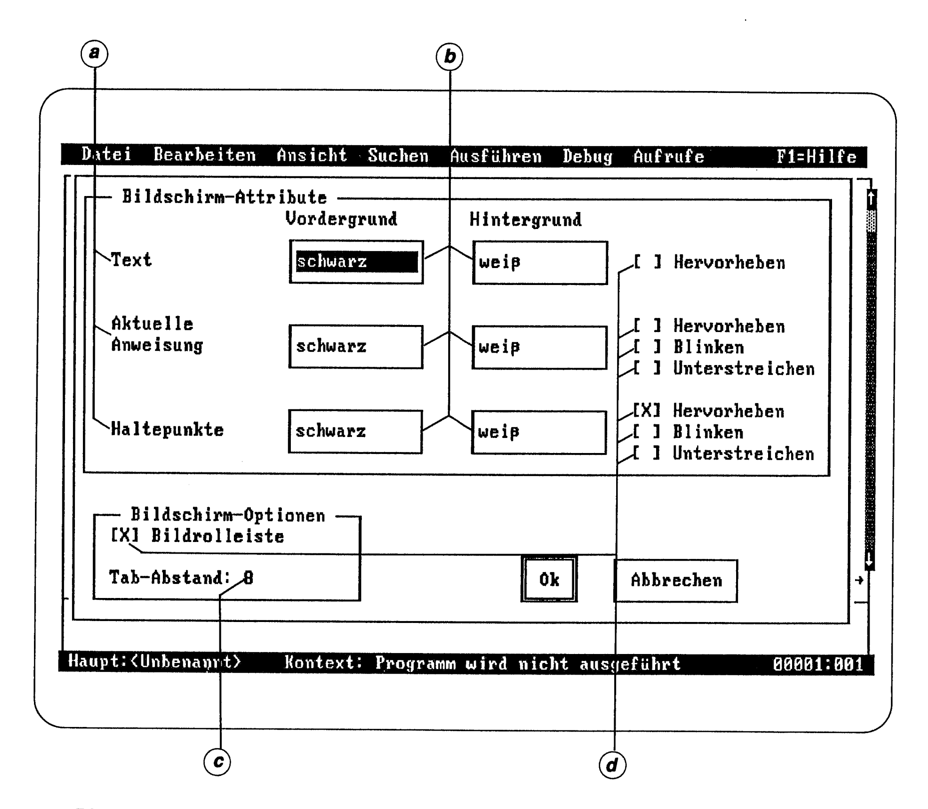
e) Verwenden Sie die Rolleiste des Listenfeldes, um mit der Maus horizontal zu rollen, wenn das Listenfeld voll ist.

f) Klicken Sie mit der Maus (oder betätigen Sie die EINGABETASTE), um die gewählten Optionen zu bestätigen.

g) Befehlsfelder

h) Klicken Sie mit der Maus (oder betätigen Sie ESC), um das gesamte Dialogfeld zu löschen.

*Abbildung 3.5 Dialogfeld **Optionen***



a) Dieser Text beschreibt den Teil der Anzeige, der betroffen ist.

b) Wahlfelder. Bewegen Sie sich mit den RICHTUNGSTASTEN durch die angebotenen Wahlmöglichkeiten.

c) Geben Sie hier eine Zahl ein, um die Einrückung zu setzen.

d) Kontrollfelder. Betätigen Sie die LEERTASTE, um eine oder alle aufgeführten Positionen ein- oder auszuschalten.

Wenn Sie mit QuickBASIC arbeiten, müssen Sie das Objekt markieren, auf das sich der nächste Befehl oder die nächste Aktion auswirken soll. Das markierte Objekt ist normalerweise invers hervorgehoben.

Der Bereich des Dialogfeldes, auf den sich Ihre nächste Aktion auswirkt, ist im Besitz der "Eingabefixierung". Der blinkende Cursor zeigt an, wo sich die Eingabefixierung befindet. Um die Eingabefixierung innerhalb eines Dialogfeldes zu bewegen

- betätigen Sie die TAB-TASTE, oder halten Sie die ALT-TASTE gedrückt, während Sie die Taste betätigen, die dem in der auszuwählenden Position hervorgehobenen Buchstaben entspricht.

  Zum Beispiel können Sie in dem Dialogfeld **Datei laden** aus dem Menü **Datei** die Option für das Laden einer Datei ("Modul", "Include" oder "Dokument") wählen, indem Sie die ALT-TASTE gedrückt halten, während Sie entweder M, I oder D betätigen.

  Beachten Sie, daß Sie bei Dialogfeldern die ALT-TASTE gedrückt halten müssen, während Sie den hervorgehobenen Buchstaben betätigen. Dies unterscheidet sich leicht von Menübefehlen, die es nicht erfordern, die ALT-TASTE gedrückt zu halten, während Sie den hervorgehobenen Buchstaben betätigen.

Die folgende Liste zeigt, welche Bedeutung es für verschiedene Teile eines Dialogfeldes hat, wenn diese sich im Besitz der Eingabefixierung befinden:

Eingabefixierung	***Bedeutung***
Textfeld	Zeigt, wo eingegebener Text erscheinen wird.
Optionsfeld	Zeigt die aktuell gewählte Option.
Kontrollfeld	Schaltet eine Option ein oder aus. Betätigen Sie die LEERTASTE, oder halten Sie die ALT-TASTE gedrückt, während Sie den hervorgehobenen Buchstaben betätigen. Wenn eine Option eingeschaltet ist, erscheint in dem Optionsfeld ein X; wenn sie ausgeschaltet ist, ist das Kontrollfeld leer.
Optionsfläche	Schaltet eine Option ein oder aus. Betätigen Sie die RICHTUNGSTASTEN, oder halten Sie die ALT-TASTE gedrückt, während Sie den hervorgehobenen Buchstaben betätigen.
Befehlsfläche	Aktiviert einen Befehl. Betätigen Sie entweder die LEERTASTE oder die EINGABETASTE. Beachten Sie, daß jedesmal, wenn Sie die EINGABETASTE betätigen, die aktuell doppelt umrandete Befehlsfläche wirksam wird, so daß keine Notwendigkeit besteht, sich zu der OK-Fläche zu bewegen, wenn Sie mit einem Dialogfeld fertig sind.

Hinweis Das Klicken auf eine Position mit der Maus hat Auswirkungen auf die Position, unabhängig davon, wo sich die Eingabefixierung befindet.

Doppelklicken auf eine Position in einem Listenfeld wählt diese Position nicht nur aus, sondern bestätigt alle vorangegangenen Angaben und schließt das Dialogfeld.

Wenn sich ein Dialogfeld öffnet, enthält es häufig Informationen von früher gewählten Optionen. Zum Beispiel öffnet sich das Dialogfeld **Optionen** mit den Optionen, die bei der letzten Änderung gesetzt wurden.

3.4 Wie Sie Fenster verwenden

Dieser Abschnitt stellt die Fenster von QuickBASIC vor und zeigt Ihnen, wie Sie

- das aktive Fenster wechseln.
- unterschiedliche Teile eines Programms in ein aktives Fenster bzw. aus einem aktiven Fenster heraus bewegen.
- die Größe eines Fensters verändern.
- Text in einem Fenster rollen.

Eine komplette Liste der Befehle zur Fenstersteuerung finden Sie in Abschnitt D.5 im Anhang D, "Zusammenfassung der Befehle".

3.4.1 Fenstertypen

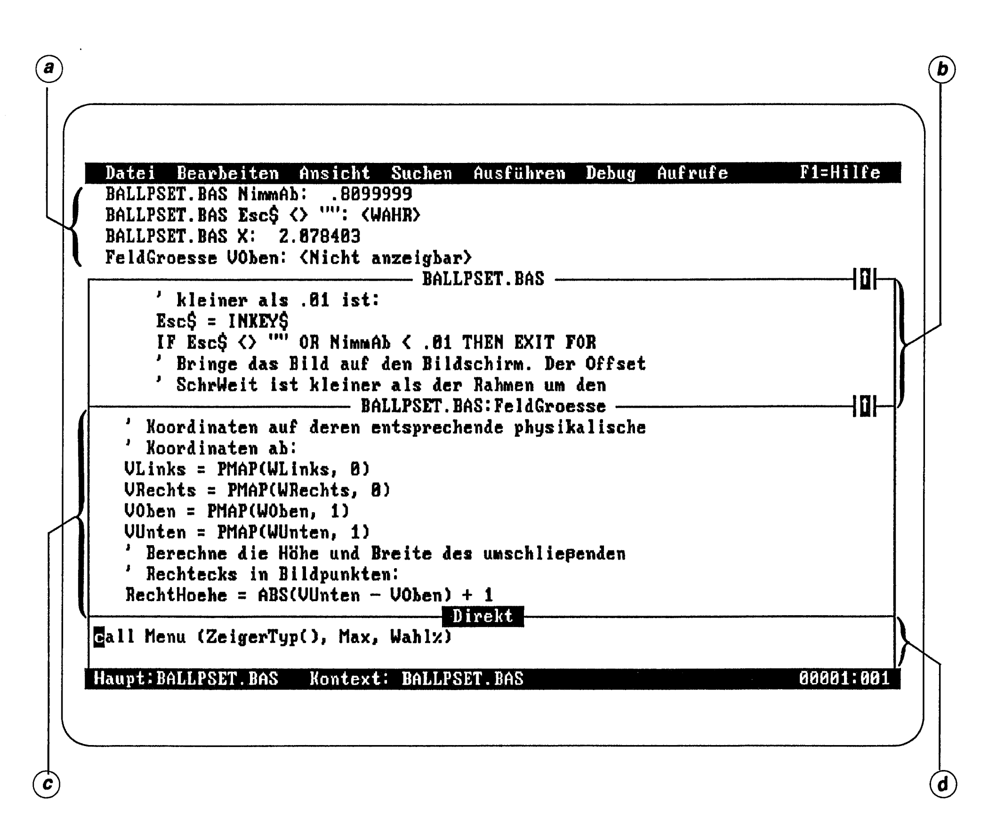
Die folgende Liste beschreibt die Fenstertypen von QuickBASIC:

Fenster	***Beschreibung***
Arbeitsbereich	Das Fenster, das auf dem Bildschirm erscheint, wenn Sie QuickBASIC starten (siehe Abbildung 3.1). Wenn Sie ein Programm laden, erscheint der Code, der sich außerhalb von **FUNCTION**- oder **SUB**-Prozeduren befindet (als Modul-Ebenen-Code bezeichnet) im Arbeitsbereich. Das Menü **Ansicht** enthält Befehle, mit denen Sie unterschiedliche Teile Ihres Programms leicht in Arbeitsbereiche hinein sowie aus Arbeitsbereichen heraus bewegen können. Um zum Beispiel eine **FUNCTION**- oder **SUB**-Prozedur in dem Arbeitsbereich sehen zu können, wählen Sie den Befehl **SUBs** aus dem Menü **Ansicht** und wählen anschließend die Prozedur aus dem Dialogfeld.

Fenster	***Beschreibung***
Direkt-Fenster	Das Fenster, das im unteren Bereich des Bildschirms erscheint, wenn Sie QuickBASIC starten. (Weitere Informationen zu dem Direkt-Fenster finden Sie in den Abschnitten 3.4.6 und 6.1.5.)
Debug-Fenster	Das Fenster, das Sie im oberen Bereich des Bildschirms öffnen können, wenn Sie debuggen. Es zeigt Variablenwerte an, während Ihr Programm läuft. (Weitere Informationen zur Verwendung des Debug-Fensters finden Sie in Abschnitt 7.3.4.1.)

Da der Arbeitsbereich zweigeteilt sein kann, können Sie insgesamt vier gleichzeitig geöffnete Fenster haben, wie in Abbildung 3.6 gezeigt.

Abbildung 3.6 Bildschirm mit vier geöffneten Fenstern



a) Debug-Fenster
b) Arbeitsbereich (oberer)
c) Arbeitsbereich (unterer)
d) Direkt-Fenster

3.4.2 Wie Sie den Arbeitsbereich teilen

Sie können den Arbeitsbereich horizontal teilen. Dies erlaubt es Ihnen, zwei Teile eines Programmes gleichzeitig zu betrachten oder zu bearbeiten.

Um den Arbeitsbereich zu teilen

1. Betätigen Sie ALT+N, um das Menü **Ansicht** zu öffnen.
2. Betätigen Sie T (der hervorgehobene Buchstabe im Befehl **Teilen**).

Der Bildschirm ist nun in zwei Teile geteilt. Sie können den Bildschirm auf die folgenden drei Arten manipulieren:

1. Betätigen Sie F6, um sich aus der oberen Hälfte in die untere Hälfte zu bewegen.
2. Betätigen Sie UMSCHALTTASTE+F6, um sich von unten nach oben zu bewegen.
3. Betätigen Sie ALT+N und anschließend T, um den alten Bildschirm wiederherzustellen. Dies löscht das untere Fenster.

3.4.3 Wie Sie das aktive Fenster wechseln

Das Fenster, das den Cursor enthält, wird als das aktive Fenster bezeichnet; das heißt, dies ist die Stelle, an der per Tastatur eingegebener Text erscheint.

Um ein anderes Fenster aktiv werden zu lassen

- Betätigen Sie F6 oder UMSCHALTTASTE+F6.

  Die Verwendung von F6 gestattet es Ihnen, sich nach unten durch die Fenster auf dem Bildschirm zu bewegen, wobei jedes Fenster nacheinander aktiv wird; UMSCHALTTASTE+F6 gestattet es Ihnen, sich durch die Fenster nach oben zu bewegen.

3.4.4 Wie Sie die Fenstergröße verändern

Sie können die Größe eines Fensters zeilenweise vergrößern oder verkleinern oder es so ausdehnen, daß es den gesamten Bildschirm ausfüllt. Um die Größe eines Fensters zu verändern, müssen Sie es zunächst zum aktiven Fenster machen. Verwenden Sie anschließend die folgenden Tastenkombinationen, um seine Größe zu vergrößern oder zu verkleinern (halten Sie die erste Taste gedrückt, während Sie die zweite betätigen).

Tastenkombination	*Ergebnis*
ALT+PLUS	Vergrößert das aktive Fenster um eine Zeile.
ALT+MINUS	Verkleinert das aktive Fenster um eine Zeile.
STRG+F10	Vergrößert das aktive Fenster so, daß es den gesamten Bildschirm ausfüllt, oder bringt es auf seine vorhergehende Größe zurück, wenn es bereits den ganzen Bildschirm beanspruchte. Falls der Arbeitsbereich zweigeteilt ist, wenn Sie STRG+F10 betätigen, stellt die nochmalige Betätigung von STRG+F10 beide Fenster auf dem Bildschirm wieder her.

3.4.5 Wie Sie in dem aktiven Fenster rollen

Häufig ist der Teil einer Datei, den Sie bearbeiten möchten zu breit oder zu lang, um in die Grenzen eines Arbeitsbereiches zu passen. Um sich die Teile, die nicht hineinpassen, anzuschauen, können Sie den Text nach oben, nach unten, nach rechts oder links bewegen. Dieser Vorgang wird als "Rollen" bezeichnet.

Sobald Sie den Rand eines Fensters erreicht haben, betätigen Sie die entsprechende RICHTUNGSTASTE, um mit dem Rollen zu beginnen. Um zum Beispiel Zeichen für Zeichen nach rechts zu rollen, gehen Sie auf das äußerst rechte Zeichen auf dem Bildschirm und halten die NACH RECHTS (→)-Taste gedrückt.

Die folgende Tabelle gibt Ihnen Auskunft darüber, wie Sie mehr als ein Zeichen gleichzeitig rollen. Die rechte Spalte dieser Tabelle zeigt Tastenkombinationen, die Sie vielleicht bequemer finden, wenn Sie Befehle im Stil von WordStar bevorzugen.

Rollfunktion	*Tastenkombinationen*	*WordStar-Entsprechungen*
Zeilenanfang	POS1	STRG+Q S
Zeilenende	ENDE	STRG+Q D
Seite nach oben	BILD ↑	STRG+R
Seite nach unten	BILD ↓	STRG+C
Ein Fenster nach links	STRG+NACH LINKS	---
Ein Fenster nach rechts	STRG+NACH RECHTS	---
Dateianfang	STRG+POS1	STRG+Q R
Dateiende	STRG+ENDE	STRG+Q C

Darüber hinaus können Sie in Ihrer Datei Textmarkierungspunkte setzen und zwischen diesen während des Bearbeitens hin- und herspringen.

3.4.6 Das Direkt-Fenster

Das untere Fenster auf dem Anfangsbildschirm von QuickBASIC, das mit dem Begriff Direkt auf der Titelleiste bezeichnet wird, dient der sofortigen Ausführung von QuickBASIC-Anweisungen. Wenn Sie zum Beispiel eine **LOCATE**-Anweisung dazu verwenden möchten, den Cursor auf einem bestimmten Punkt des Ausgabebildschirms zu plazieren, lassen Sie das Direkt-Fenster aktiv werden, indem Sie F6 betätigen, und geben anschließend die folgende Zeile ein (hier ist die gewählte Zeile 4, die Spalte ist 5):

```
cls : locate 4,5 : print "Test"
```

Sobald Sie die EINGABETASTE betätigt haben, werden Sie die Ausgabe sehen. Betätigen Sie eine beliebige Taste, um in das Direkt-Fenster zurückzukehren, bearbeiten Sie die Zeile, um die Argumente Zeile und Spalte einzustellen, und wiederholen Sie diesen Vorgang, bis das Wort "Test" genau dort erscheint, wo Sie es wünschen. Wenn das Wort Test dort erscheint, wo Sie es wünschen, können Sie die Anweisung in Ihr Programm kopieren.

Sie können in das Direkt-Fenster bis zu zehn einzelne Zeilen eingeben und sich anschließend zwischen diesen mit den RICHTUNGSTASTEN bewegen. Die Zeilenlänge ist auf 256 Zeichen beschränkt. Mehrere Anweisungen auf einer Zeile sind erlaubt, aber jede vollständige Anweisung muß von der nächsten mit einem Doppelpunkt (:) getrennt werden, wie in dem obigen Beispiel gezeigt. Immer dann, wenn Sie den Cursor auf einer Zeile plazieren und die EINGABETASTE betätigen, werden nur die Anweisungen dieser Zeile ausgeführt. Weitergehende Erläuterungen zu dem Direkt-Fenster finden Sie in Abschnitt 6.1.5.

3.5 Wie Sie vorübergehend zu DOS zurückkehren (Betriebssystem)

Der Befehl **Betriebssystem** aus dem Menü **Datei** gestattet es Ihnen, vorübergehend auf die DOS-Befehlsebene zurückzukehren, auf der Sie andere Programme und DOS-Befehle ausführen können. QuickBASIC verbleibt im Speicher, so daß Sie zu derselben Stelle in Ihrem Programm zurückkehren können, ohne dieses erneut zu laden.

QuickBASIC muß die Datei COMMAND.COM finden, bevor es den Befehl Betriebssystem ausführen kann. QuickBASIC sucht nach COMMAND.COM zunächst in dem Verzeichnis, das in der Umgebungsvariablen COMSPEC angegeben ist, anschließend in dem aktuellen Verzeichnis. Weitere Informationen zu COMMAND.COM sowie COMSPEC finden Sie in Ihrer DOS-Dokumentation.

Um von der DOS-Befehlsebene zu QuickBASIC zurückzukehren

1. Geben Sie

```
exit
```

ein.

2. Betätigen Sie die EINGABETASTE.

Der QuickBASIC-Bildschirm erscheint wieder so, wie Sie Ihn verlassen haben.

3.6 Wie Sie QuickBASIC beenden (Ende)

Der Befehl **Ende** aus dem Menü **Datei** entfernt QuickBASIC aus dem Speicher und bringt Sie zurück zur DOS-Eingabeaufforderung.

Wenn Sie ein neues oder verändertes Programm beenden, das nicht gespeichert wurde, zeigt QuickBASIC ein Dialogfeld an, das Sie fragt, ob Sie das Programm speichern möchten. Verwenden Sie einen dieser drei Abläufe:

1. Um das Programm in seinem aktuellen Stadium zu speichern, betätigen Sie die EINGABETASTE.

   Wenn es sich um ein neues Programm handelt, dem Sie bis jetzt noch keinen Namen gegeben haben, zeigt QuickBASIC das Dialogfeld des Befehls **Speichern unter**. Weitere Informationen zum Speichern von Dateien finden Sie in Abschnitt 4.2.2.
2. Um zum Betriebssystem zurückzukehren, ohne die (evtl. geänderte) Datei zu speichern, betätigen Sie N oder bewegen die Eingabefixierung auf das Feld "Nein" und betätigen anschließend die EINGABETASTE oder die LEERTASTE.
3. Um zur QuickBASIC-Umgebung zurückzukehren (das heißt, doch nicht beenden), betätigen Sie die ESC-TASTE.

4 Wie Sie Quelldateien verwalten

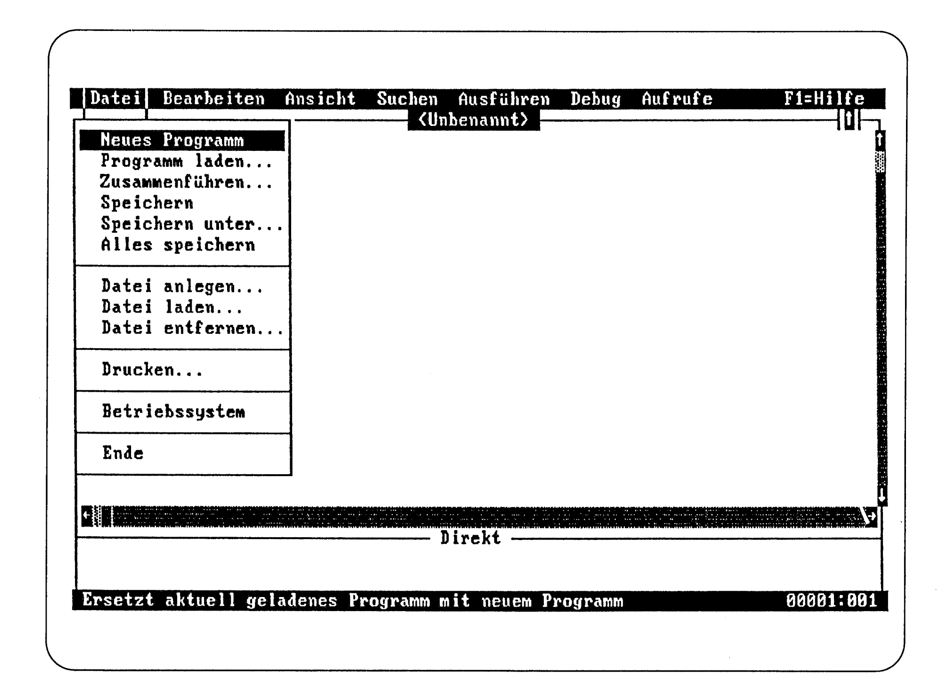
Dieses Kapitel erläutert, wie in der QuickBASIC-Umgebung Dateien manipuliert werden. Nachdem Sie dieses Kapitel gelesen haben, werden Sie die Unterschiede zwischen Programmen, Modulen, Include-Dateien und Dokumenten kennen. Darüber hinaus werden Sie wissen, wie Sie folgendes durchführen können:

- Programme laden, listen und speichern.
- Module erstellen.
- Den Unterschied zwischen dem Hauptmodul eines Programms und seinen weiteren Modulen verstehen.
- Module eines Programms laden bzw. entfernen.
- Include-Dateien betrachten und bearbeiten.
- Dokumente in QuickBASIC laden, um diese zu betrachten und zu bearbeiten.
- Die Inhalte zweier Dateien mischen.
- Dateien drucken.

4.1 Wie Sie QuickBASIC-Dateien klassifizieren

Das in Abbildung 4.1 gezeigte Menü **Datei** ist Ihr zentraler Punkt zur Arbeit mit Dateien in QuickBASIC. Mit dem Menü **Datei** können Sie neue Dateien erstellen, vorhandene Dateien in QuickBASIC laden, Dateien verändern oder Dateien aus Ihrem Programm entfernen.

*Abbildung 4.1 Das Menü **Datei***



Was Sie mit einer Datei durchführen können, hängt weitestgehend davon ab, wie Sie die Datei laden oder anlegen. QuickBASIC kennt vier unterschiedliche Dateiarten, und die folgende Liste beschreibt kurz jede dieser vier Dateiarten (Einzelheiten zur Arbeit mit jeder Dateiart finden Sie in den Abschnitten 4.2 bis 4.6):

- **Programm**

  Ein Programm enthält eine oder mehrere BASIC-Anweisungen, die von QuickBASIC in Anweisungen für Ihren Computer übersetzt werden.

  Wenn Sie QuickBASIC gestartet haben und damit beginnen, im Arbeitsbereich Eingaben vorzunehmen, geht QuickBASIC davon aus, daß Sie ihm Programmanweisungen eingeben, so daß es die Eigenschaften seines "intelligenten" Editors einschaltet. Wenn der intelligente Editor eingeschaltet ist, wird jede Textzeile auf die richtige BASIC-Syntax überprüft, sobald Sie die EINGABETASTE betätigen, und Sie erhalten eine Fehlermeldung, wenn Sie einen Fehler gemacht haben. Der intelligente Editor wird ebenfalls eingeschaltet, wenn Sie aus dem Menü **Datei Neues Programm** oder **Programm laden** wählen, oder wenn Sie QuickBASIC mit dem Namen eines Programms starten, das Sie bearbeiten möchten.

Wenn Sie hauptsächlich in BASICA programmiert haben, sind Sie wahrscheinlich daran gewöhnt, sich ein Programm als gleichbedeutend mit einer Datei vorzustellen. Wenn Ihre Programme normalerweise klein und einfach sind, können Sie mit QuickBASIC fortfahren, in dieser Art zu programmieren. Weitere Informationen zum Laden, Listen und Speichern von Programmen finden Sie in Abschnitt 4.2.

- **Modul**

  Ein Modul ist eine einzelne, selbständige Programmkomponente. Ein Programm besteht immer aus mindestens einem Modul. Wenn Sie ein Programm aus mehreren Modulen erstellen, wird jedes Modul auf Diskette als eine separate Datei gespeichert, wenn Sie das Programm abspeichern. Module sind praktisch, wenn Sie längere Programme haben, oder wenn Sie dieselben Codezeilen immer wieder in unterschiedlichen Programmen einsetzen.

  Sie sind nicht gezwungen, die QuickBASIC-Eigenschaften für mehrmodulige Programme zu verwenden, wenn Sie ein Programm schreiben. Sie können nach wie vor alles in eine Datei schreiben, und wenn Sie das Programm abspeichern, wird es als eine Datei gespeichert. Das Aufbrechen großer Programme in einzelne Module erleichtert es Ihnen jedoch erheblich, häufig benötigte Prozeduren in vielen verschiedenen Programmen gemeinsam zu benutzen. Mehrmodulige Programme dürfen darüber hinaus viel größer sein, da jedes Modul 64K haben kann.

  Nachdem ein Modul einmal fehlerfrei vorliegt, können Sie es als Baustein in vielen verschiedenen Programmen verwenden. Wenn Sie dieselben **SUB**- und **FUNCTION**-Prozeduren in verschiedenen Programmen verwenden, können Sie diese Prozeduren in ein gemeinsames Modul schreiben und dieses Modul anschließend bei Bedarf in Ihre Programme laden. (Außerdem ist es ein kleiner, leichter Schritt, solche Module in eine Quick-Bibliothek zu überführen, so daß Sie diese Module nicht jedesmal, wenn Sie deren Prozeduren aufrufen möchten, laden müssen. Weitere Informationen finden Sie in Kapitel 8, "Quick-Bibliotheken".)

  Um mehr über die Verwendung von Modulen zu erfahren, schlagen Sie in den Abschnitten 4.3 bis 4.4 sowie Kapitel 6, "Wie Sie QuickBASIC-Programme erstellen und ausführen", nach.

- **Include**

  Eine "Include"-Datei ist eine Datei, deren Anweisungen in Ihr Programm kompiliert werden, wenn QuickBASIC auf den Dateinamen trifft, der einem **$INCLUDE**-Metabefehl folgt. Zum Beispiel nimmt die folgende Zeile den Inhalt der Include-Datei DEKLA.BI auf:

  ```
  ' $INCLUDE: 'DEKLA.BI'
  ```

  Als Include-Dateien geöffnete Dateien sind der Syntaxüberprüfung und Formatierung unterworfen. Beachten Sie, daß QuickBASIC dem Inhalt von Include-Dateien einige Einschränkungen auferlegt.

  Weitere Informationen zu Include-Dateien finden Sie in Abschnitt 4.5.

- **Dokument**

  Eine als Dokument geöffnete Datei kann in QuickBASIC genauso wie mit einem normalen Textverarbeitungsprogramm bearbeitet werden. QuickBASIC betrachtet ein Dokument nicht als BASIC-Programm, so daß es Dokumente weder auf Syntax überprüft noch deren Text formatiert (zum Beispiel BASIC-Schlüsselworte in Großbuchstaben umwandelt). Wenn Sie ein mehrmoduliges Programm als Dokument laden, wird nur das Hauptmodul geladen (weitere Informationen finden Sie in Abschnitt 4.4.1, "Die Datei .MAK").

Hinweis Nachdem Sie eine der vorhergehenden Dateiarten in QuickBASIC geladen haben, können Sie den Dateinamen in dem Listenfeld des Dialogfeldes **SUBs** aus dem Menü **Ansicht** sehen.

Während in diesem Listenfeld ein Dateiname erscheint, können Sie die Datei nicht erneut laden, auch nicht mit einer anderen Klassifizierung. Wenn Sie zum Beispiel eine Datei als Dokument geladen haben, können Sie diese solange nicht erneut als Include-Datei laden, bis Sie die als Dokument klassifizierte Kopie entfernt haben.

4.2 Programme

Wie bereits in Abschnitt 4.1, "Wie Sie QuickBASIC-Dateien klassifizieren", erwähnt, arbeiten Sie automatisch mit einem Programm, wenn Sie beginnen, Text in den Arbeitsbereich einzugeben. Um das aktuelle Programm zu entfernen und ein ganz neues zu beginnen, wählen Sie den Befehl **Neues Programm** aus dem Menü **Datei**. Um ein bereits existierendes Programm zur Ansicht oder zum Bearbeiten zu öffnen, wählen Sie den Befehl **Programm laden** aus dem Menü **Datei**. Beide dieser Anweisungen löschen alle zuvor geladenen Dateien.

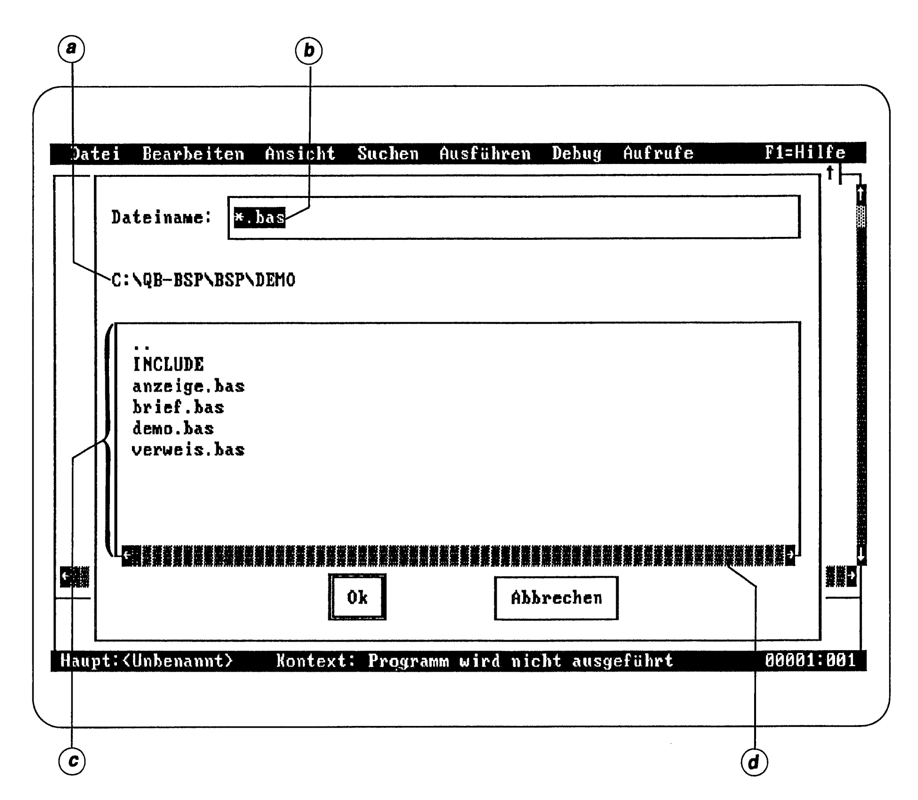
4.2.1 Wie Sie Programme laden und listen (Programm laden)

Der Befehl **Programm laden** aus dem Menü **Datei** lädt ein Programm von Diskette in den Speicher. Wenn ein Programm Include-Dateien verwendet, werden die Include-Dateien kompiliert, aber nicht angezeigt. (Weitere Informationen zum Anzeigen und Bearbeiten von Include-Dateien finden Sie in Abschnitt 4.5.) Wenn ein Programm aus mehreren Modulen besteht, werden alle zu dem Programm gehörenden Module geladen.

Sie können den Befehl **Programm laden** auch dazu verwenden, die Dateien und Verzeichnisse Ihres Systems aufzulisten (siehe Abschnitt 4.2.1.1, "Wie Sie eine Datei angeben").

Wenn Sie den Befehl **Programm laden** aus dem Menü **Datei** wählen, öffnet sich das in Abbildung 4.2 gezeigte Dialogfeld:

Abbildung 4.2 Das Dialogfeld Programm laden



a) Verzeichnis der gezeigten Dateien

b) Nehmen Sie hier Eingaben vor, um Dateilisten anzuzeigen oder eine bestimmte Datei zu laden.

c) Das Listenfeld zeigt die Namen der Unterverzeichnisse und Dateien des aktuellen Verzeichnisses an.

d) Verwenden Sie diese Rolleiste mit der Maus, um die Liste horizontal zu rollen.

Verzeichnis- und Dateinamen werden in dem Listenfeld in Spalten angezeigt. Verzeichnisnamen erscheinen in Großbuchstaben oben auf der Liste. Wenn nicht alle diese Namen in das Feld passen, verwenden Sie eine der RICHTUNGSTASTEN, um den Cursor an den Rand des Feldes zu bewegen, und halten Sie diese gedrückt, um die Liste zu rollen. Beachten Sie, daß Sie das Listenfeld nur nach links oder rechts rollen können.

4.2.1.1 Wie Sie eine Datei angeben

Das Dialogfeld **Programm laden** bietet zwei Möglichkeiten, die Datei anzugeben, die Sie laden möchten.

1. Die erste Methode lautet wie folgt:
   - Geben Sie den Programmnamen in das Textfeld ein.

     QuickBASIC nimmt die Erweiterung .BAS für alle Dateinamen an, die keine Erweiterung haben. Wenn Sie eine Datei laden möchten, die keine Erweiterung hat, geben Sie direkt hinter dem Dateinamen einen Punkt (.) ein.

2. Die zweite Methode zur Angabe einer Datei ist wie folgt:

   a. Betätigen Sie die TAB-TASTE, um sich in das Listenfeld zu bewegen, verwenden Sie anschließend eine der folgenden Techniken, um sich durch das Listenfeld zu bewegen, bis die Datei, die Sie laden möchten, hervorgehoben ist:

   – Verwenden Sie die RICHTUNGSTASTEN.

   – Betätigen Sie den ersten Buchstaben des Namens der Datei, die Sie laden möchten. Wenn sich zum Beispiel die Dateien

   ```
   PROG1.BAS
   PROG2.BAS
   PROG3.BAS
   PROG4.BAS
   ```

   in dem aktuellen Verzeichnis befinden, betätigen Sie P dreimal, um die Eingabefixierung auf PROG3.BAS zu bewegen.

   b. Betätigen Sie die EINGABETASTE.

   Der Programmtext erscheint in einem Fenster mit dem Dateinamen auf der Titelleiste und steht für Sie bereit, bearbeitet oder gestartet zu werden.

Hinweis Um ein Programm mit der Maus zu laden, doppelklicken Sie den Dateinamen in dem Listenfeld.

4.2.1.2 Wie Sie Verzeichnisinhalte auflisten

Verwenden Sie das Dialogfeld **Programm laden**, um den Inhalt eines Verzeichnisses des Systems aufzulisten. Wenn Sie einen Verzeichnisnamen in das Textfeld eingeben oder einen Verzeichnisnamen in dem Listenfeld wählen, listet QuickBASIC alle Dateien und Unterverzeichnisse des von Ihnen gewählten Verzeichnisses auf. (Wenn Sie einen Dateinamen eingeben oder wählen, wird diese Datei dagegen in den Speicher geladen.) Das Listenfeld zeigt Dateinamen in Kleinbuchstaben und Verzeichnisnamen in Großbuchstaben an.

Sie können die DOS-Jokerzeichen (? und *) verwenden, um eine Gruppe von Dateien aufzulisten, wie in den folgenden Beispielen gezeigt:

Aufgabe	*Durchführung*
Alle Dateien des aktuellen Verzeichnisses auflisten	Geben Sie `*.*` in das Textfeld ein.
Die Dateien des Hauptverzeichnisses von Laufwerk A auflisten	Geben Sie `A:\*.*` in das Textfeld ein.
Alle Dateien in einem \BIN benannten Unterverzeichnis auflisten	Bewegen Sie in dem Listenfeld die Hervorhebung auf BIN, betätigen Sie die LEERTASTE.
Alle Dateien des aktuellen Verzeichnisses auflisten, die die Erweiterung .BI haben	Geben Sie `*.BI` in das Textfeld ein.
Alle Dateien des Verzeichnisses auflisten, das direkt über dem aktuellen Verzeichnis liegt	Geben Sie `..` in das Textfeld ein, oder wenn Sie sich bereits im Listenfeld befinden, betätigen Sie eine der RICHTUNGSTASTEN oder den Punkt (.), um `..` zu wählen. Wenn `..\*` gefolgt von einer Erweiterung erscheint, betätigen Sie die EINGABETASTE.

Hinweis Das "aktuelle Arbeitsverzeichnis" ist das von dem DOS-Befehl CD gezeigte Verzeichnis. Wenn Sie aus einem Listenfeld heraus ein Verzeichnis wählen, listen Sie im Gegensatz dazu nur die Dateien auf, die das Verzeichnis enthält, und wechseln nicht das aktuelle Arbeitsverzeichnis auf DOS-Ebene. Wenn Sie das aktuelle Arbeitsverzeichnis wechseln möchten, machen Sie das Direkt-Fenster aktiv, und verwenden Sie die Anweisung **CHDIR.**

4.2.2 Wie Sie ein Programm speichern (Speichern, Alles speichern, Speichern unter)

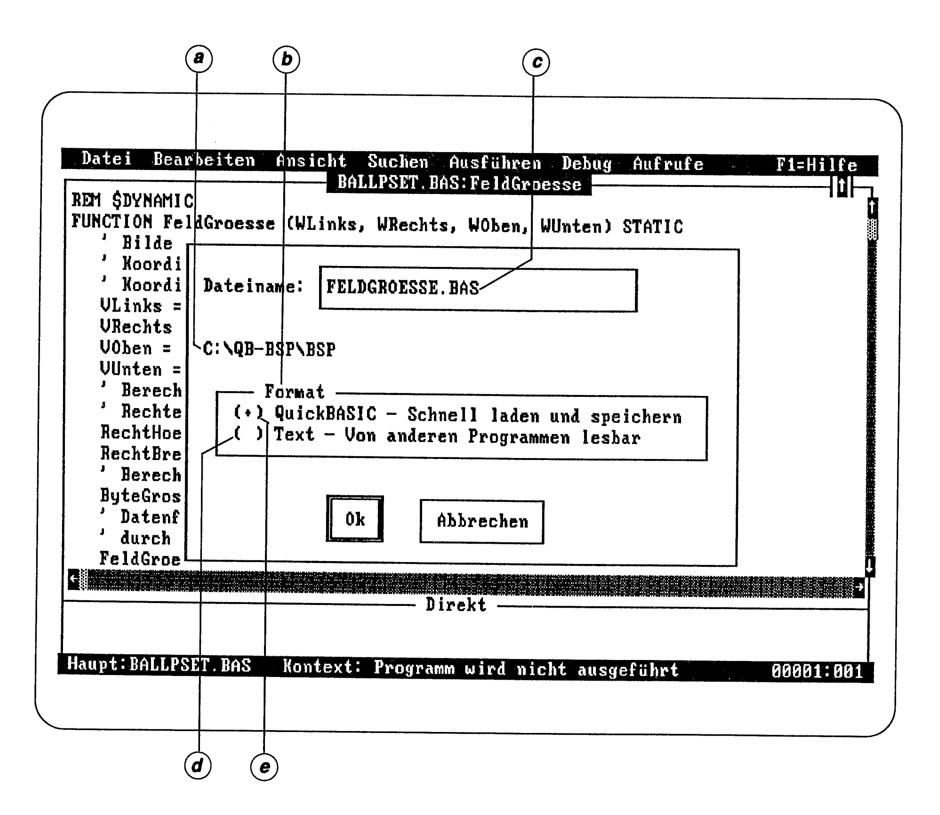
Jeder der Befehle **Speichern, Alles speichern** und **Speichern unter** speichert Dateien. Die folgende Liste erläutert die Unterschiede zwischen diesen Befehlen:

Befehl	*Ergebnis*
Speichern	Schreibt den Inhalt des aktuellen Moduls (die Datei, die im aktiven Fenster angezeigt wird) auf eine Diskettendatei.
Alles speichern	Speichert alle aktuell geladenen Dateien, die seit der letzten Speicherung verändert wurden. Eine aktuell geladene Datei ist eine Datei, deren Name in dem Dialogfeld des Befehls **SUBs** aus dem Menü **Ansicht** erscheint. **Alles speichern** speichert alle veränderten Dateien und ist hilfreich, wenn Sie mit mehreren Modulen arbeiten.

Befehl	***Ergebnis***
Speichern unter	Speichert das aktuelle Modul unter dem Namen, den Sie angeben. Dies ist hilfreich zum Speichern eines Moduls unter einem neuen Namen sowie zur Veränderung des Formats, in dem die Datei gespeichert wird. Wenn Sie den Namen wechseln, existiert die alte Datei noch mit dem Namen, den sie bei der letzten Speicherung hatte.

Wenn Sie **Speichern unter** aus dem Menü **Datei** wählen, öffnet sich das in Abbildung 4.3 gezeigte Dialogfeld:

*Abbildung 4.3 Das Dialogfeld **Speichern unter***



a) Standardverzeichnis zum Speichern
b) Diese Optionen ermöglichen Ihnen festzulegen, wie Ihr Programm gespeichert werden soll.
c) Geben Sie hier den neuen Dateinamen ein.
d) Es wird im ASCII-Format gespeichert.
e) Es wird im QuickBASIC-Schnellade-Format gespeichert.

Die folgende Liste erläutert das Dialogfeld:

Option	*Funktion*
Textfeld Dateiname	Enthält den Namen der Datei, die Sie speichern möchten. Wenn Sie eine Datei speichern, die bereits einen Namen hat, dann erscheint der Dateiname in dem Textfeld markiert. Wenn Sie die Datei mit einem anderen Namen speichern möchten, beginnen Sie mit der Eingabe. Ihre Eingabe ersetzt den alten Namen. Wenn Sie die Datei das nächstemal speichern, erscheint in dem Textfeld der neue Dateiname.
Option QuickBASIC – Schnell laden und speichern	Speichert Ihr Programm im QuickBASIC-Format. Ein Programm, das auf diese Art gespeichert ist, wird schneller geladen als ein Programm, das als Textdatei (siehe unten) gespeichert ist, kann aber nur mit QuickBASIC bearbeitet werden. Da die Datei keine ASCII-Datei ist, werden Sie nicht in der Lage sein, sie mit einem anderen Texteditor zu verändern. Wenn Sie nicht explizit eine Option wählen, ist dies die Art, in der QuickBASIC neue Programme speichert. Dokumente und Include-Dateien können nicht in diesem Format gespeichert werden.
Option Text – Von anderen Programmen lesbar	Speichert Ihre Datei als ASCII- oder Textdatei auf Diskette. Textdateien können von jedem beliebigen Texteditor oder Textverarbeitungsprogramm, der bzw. das ASCII-Dateien liest, gelesen werden. Dateien, die Sie mit der Option "Dokument" oder "Include" klassifizieren, werden immer in diesem Format gespeichert.

Das in Abbildung 4.3 gezeigte Dialogfeld erscheint ebenfalls, wenn Sie einen der folgenden Schritte durchführen:

- **Speichern** oder **Alles speichern** wählen, während sich ein unbenanntes Programm im Speicher befindet.
- Ein weiteres Programm laden, während sich ein unbenanntes Programm im Speicher befindet.
- Eine ausführbare Datei oder Quick-Bibliothek erstellen, nachdem das aktuelle Programm geändert wurde.

4.3 Module

Wenn QuickBASIC ein Programm lädt, wird jede von ihm geladene Datei als Modul bezeichnet. Ein Programm kann viele Module haben, aber jedes Programm besteht aus mindestens einem Modul, das in QuickBASIC als das "Hauptmodul" bezeichnet wird. Include-Dateien sind keine Module, obwohl sie in einem Programm eine wichtige Rolle spielen können.

Wenn Sie eine Datei als ein Modul laden oder erstellen, formatiert QuickBASIC diese und überprüft deren Syntax.

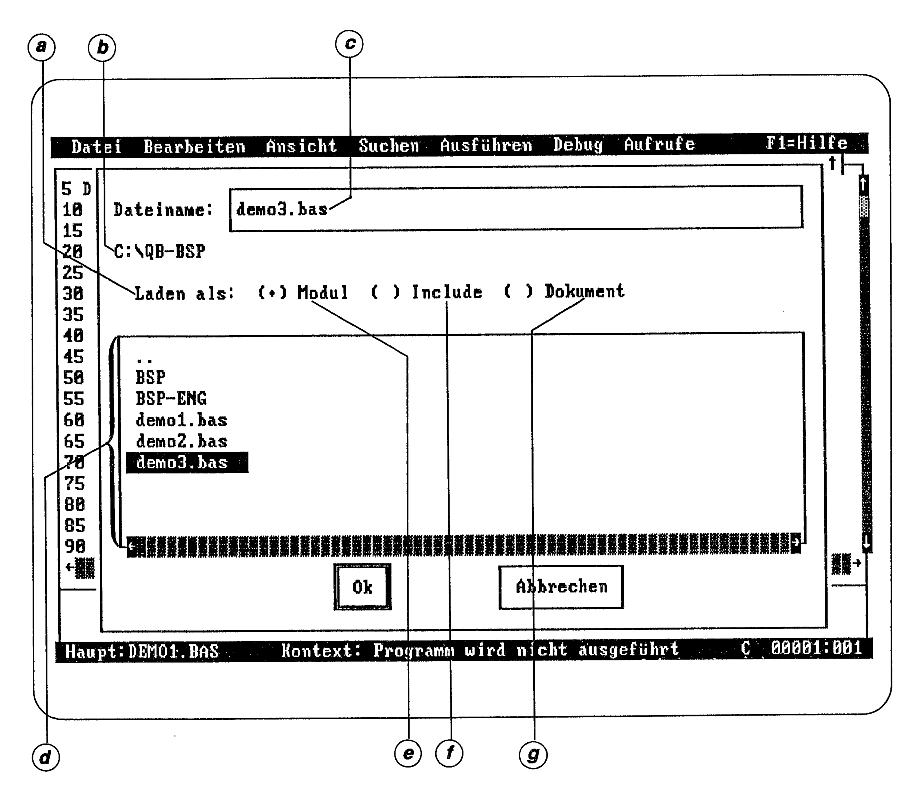
Um eine Liste der aktuell im Speicher befindlichen Module zu erhalten, wählen Sie den Befehl **SUBs** aus dem Menü **Ansicht**. Die Dateinamen erscheinen in Großbuchstaben im Dialogfeld **SUBs**.

4.3.1 Wie Sie Module erstellen und laden (Datei laden)

Der Befehl **Datei laden** aus dem Menü **Datei** lädt einzelne Module – sowie Include-Dateien und Dokumentdateien – von Diskette/Festplatte in den Speicher.

Wenn Sie **Datei laden** aus dem Menü **Datei** wählen, erscheint das in Abbildung 4.4 gezeigte Dialogfeld.

Abbildung 4.4 Das Dialogfeld ***Datei laden***



a) Es ist wichtig, bei der Erstellung einer Datei die richtige Option zu setzen.

b) Verzeichnis der gezeigten Dateien

c) Nehmen Sie hier Eingaben vor, um eine Liste der Dateien anzuzeigen oder eine bestimmte Datei zu laden.

d) Das Listenfeld zeigt die Namen der Dateien und Unterverzeichnisse des aktuellen Verzeichnisses an.

e) Es wird ein neues Modul geladen; alle anderen geladenen Dateien verbleiben im Speicher.

f) Es wird eine Include-Datei geladen; alle anderen geladenen Dateien verbleiben im Speicher.

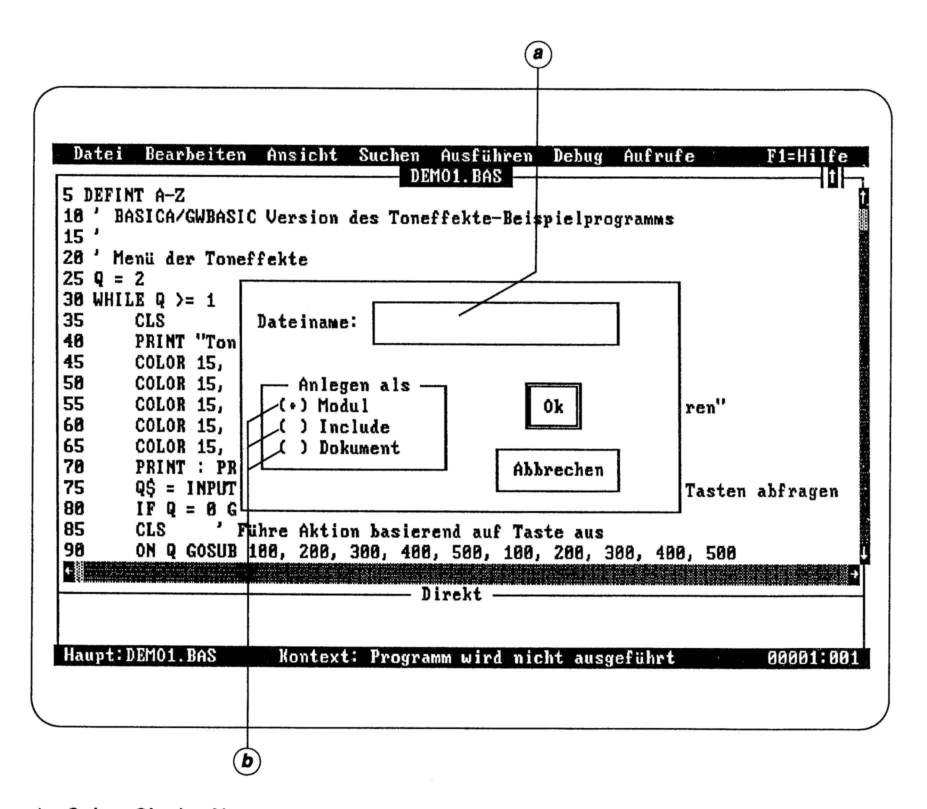
g) Es wird eine Textdatei geladen; alle anderen geladenen Dateien verbleiben im Speicher.

Beachten Sie, daß die Befehle **Neues Programm** und **Programm laden** aus dem Menü **Datei** Dateien ebenfalls als Module laden, da QuickBASIC ein Programm als aus einem oder mehreren Modulen bestehend betrachtet. Im Falle eines einmoduligen Programmes erstellen oder laden diese Befehle nur ein Hauptmodul. Bei einem mehrmoduligen Programm lädt **Programm laden** das Hauptmodul und alle anderen Module, aus denen Ihr Programm besteht. Weitere Informationen zu dieser Technik finden Sie in Abschnitt 4.4.1, "Die Datei .MAK".

Um ein neues Modul als Teil des aktuell im Speicher befindlichen Programmes zu beginnen

1. Wählen Sie den Befehl **Datei anlegen** aus dem Menü **Datei.** Es erscheint das Dialogfeld **Datei anlegen** (siehe Abbildung 4.5).

Abbildung 4.5 Das Dialogfeld Datei anlegen



a) Geben Sie den Namen der Datei ein, die Sie anlegen möchten.
b) Es ist wichtig, beim Erstellen einer Datei die richtige Option zu setzen.

2. Geben Sie in das Textfeld des Dialogfeldes den Namen des Moduls ein, das Sie anlegen möchten. Beachten Sie, daß die Optionsfläche "Modul" schon markiert ist.
3. Betätigen Sie die EINGABETASTE, um das neue Modul zu erstellen.

Um Module aus anderen Programmen in das aktuell im Speicher befindliche Programm einzubinden

1. Wählen Sie den Befehl **Datei laden** aus dem Menü **Datei.**
2. Markieren Sie den Namen des Moduls, das die Prozeduren enthält, die Sie in dem aktuellen Programm verwenden möchten. Es gibt zwei Möglichkeiten, die Datei, die Sie laden möchten, anzugeben:
   - Geben Sie den Namen der Datei ein, betätigen Sie anschließend die EINGABETASTE.
   - Betätigen Sie die TAB-TASTE, um sich in das Listenfeld zu bewegen, und benutzen Sie anschließend die RICHTUNGSTASTEN, um sich durch das Listenfeld zu bewegen, bis die Datei, die Sie laden möchten, hervorgehoben ist. (Wenn Sie den ersten Buchstaben des Namens der Datei, die Sie laden möchten, betätigen, bewegt sich die Hervorhebung in alphabetischer Reihenfolge durch alle Dateinamen, die mit diesem Buchstaben beginnen. Wenn Sie zum Beispiel nach der Datei PROG1.BAS suchen, wird durch das Betätigen von "p" die Eingabefixierung schließlich auf die Datei mit diesem Namen bewegt.)
3. Betätigen Sie die EINGABETASTE, um die Datei als Modul zu laden, da die Option "Modul" bereits markiert ist.
4. Wiederholen Sie die Schritte 1 bis 3 für jedes Modul, das Sie laden möchten.
5. Speichern Sie das Programm ab mit dem Befehl **Alles speichern**, wenn alle Module geladen sind.

4.3.2 Wie Sie mehrere Module anzeigen lassen (SUBs)

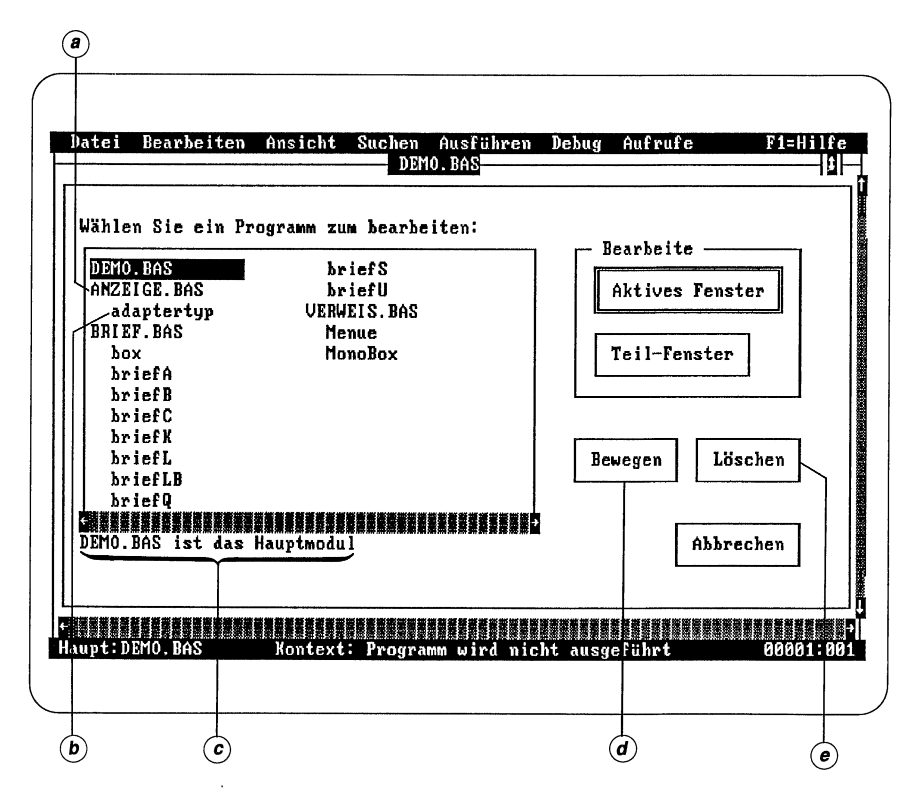
Obwohl QuickBASIC nur zwei Arbeitsbereiche zur gleichen Zeit anzeigen kann, kann es so viele Module laden, wie der Speicher fassen kann. Wenn Sie ein neues Modul in den Speicher laden, wird dieses im aktiven Fenster angezeigt. Das Modul, das sich zuvor in dem Fenster befand, wird nicht mehr angezeigt, befindet sich jedoch weiterhin im Speicher.

Um ein geladenes Modul im Arbeitsbereich anzuzeigen

1. Wählen Sie den Befehl **SUBs** aus dem Menü **Ansicht**. Es erscheint das Dialogfeld **SUBs** (siehe Abbildung 4.6).



Abbildung 4.6 Das Dialogfeld ***SUBs***



a) Die im Speicher befindlichen Dateien werden in Großbuchstaben aufgeführt.
b) Prozeduren stehen eingerückt unter den Modulen, die die Prozeduren enthalten.
c) Beschreibt Objekte, die im Listenfeld hervorgehoben sind.
d) Bewegt Prozeduren von einem in ein anderes Modul.
e) Entfernt eine Datei aus dem Speicher oder löscht eine ***SUB****- oder* ***FUNCTION****-Prozedur.*

2. Wählen Sie das Modul (oder eine bestimmte Prozedur innerhalb des Moduls), das Sie anzeigen möchten.
3. Betätigen Sie die EINGABETASTE.

Wenn Sie zwei Module oder Prozeduren betrachten möchten, wählen Sie den Befehl Fenster teilen, um den Bildschirm in zwei Arbeitsbereiche zu teilen. Das von Ihnen gewählte Modul erscheint in dem neuen Fenster.

4.3.3 Wie Sie mehrere Module speichern (Alles speichern)

Es ist am bequemsten, den Befehl **Alles speichern** aus dem Menü **Datei** zu verwenden, um Ihre Arbeit abzuspeichern, wenn Sie mit mehreren Modulen arbeiten. **Alles speichern** speichert alle Module ab, die seit der letzten Speicherung verändert wurden. Der Befehl **Speichern** speichert nur das Modul ab, das mit dem aktiven Fenster verknüpft ist.

4.4 Wie Sie das Hauptmodul wechseln (Hauptmodul bestimmen)

Jedes Programm hat mindestens ein Modul. Falls ein Programm mehrere Module besitzt, dann enthält das "Hauptmodul" die erste auszuführende Anweisung, wenn das Programm gestartet wird.

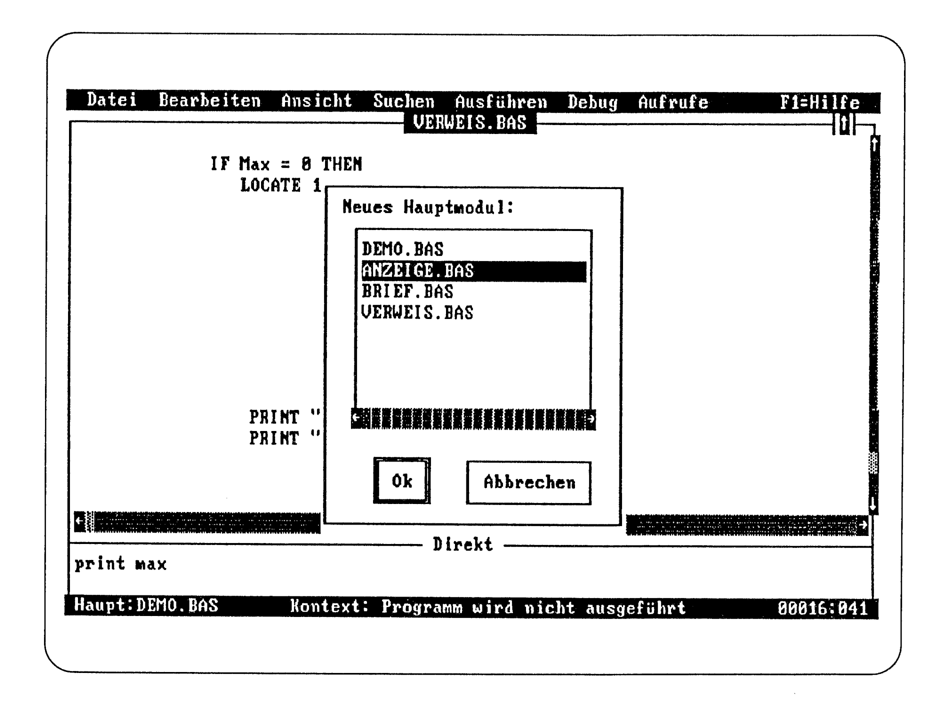
Ein Modul kann Teil vieler verschiedener Programme sein, aber es kann nur das Hauptmodul eines Programmes sein (das Programm, dessen Namen es trägt).

Die Statusleiste zeigt den Namen des Hauptmoduls des aktuell geladenen Programmes an. Das Hauptmodul ist ebenfalls der erste Eintrag in jedem Listenfeld, das die Module eines Programmes anzeigt.

Während Sie Ihr Programm bearbeiten, kann es passieren, daß Sie Ihr Programm neu aufbauen möchten und wünschen, daß dessen Ausführung mit einem anderen Modul beginnt als mit dem aktuellen Hauptmodul. Um dies durchzuführen

1. Wählen Sie den Befehl **Hauptmodul bestimmen** aus dem Menü **Ausführen**. Das Dialogfeld des Befehls **Hauptmodul bestimmen** enthält eine Liste der aktuell geladenen Module (siehe Abbildung 4.7).

Abbildung 4.7 Das Dialogfeld ***Hauptmodul bestimmen***



2. Markieren Sie das Modul, das Sie zum Hauptmodul machen möchten, und betätigen Sie die EINGABETASTE.

4.4.1 Die Datei .MAK

Wenn Sie ein mehrmoduliges Programm speichern, erstellt QuickBASIC eine spezielle Datei, die die Namen aller Module des Programms enthält. Der Name dieser Datei besteht aus dem Basisnamen des Hauptmoduls sowie der Erweiterung .MAK, deswegen der Ausdruck ".MAK-Datei". Das Hauptmodul eines Programms ist der erste Eintrag in der .MAK-Datei.

Wenn Sie ein Programm erneut laden, verwendet QuickBASIC die .MAK-Datei, um alle Module des Programms zu finden. Daher sollten Sie eine .MAK-Datei nicht löschen, und wenn Sie ein mehrmoduliges Programm in ein anderes Verzeichnis bewegen, sollten Sie auch die .MAK-Datei dorthin bewegen.

In einer .MAK-Datei befinden sich nur Programmodule. QuickBASIC fügt keine weiteren Dateitypen in die .MAK-Liste ein. Wenn sich Ihr Programm aus Include-Dateien aufbaut, muß das Programm jede dieser Dateien als ein Argument für einen Metabefehl **$INCLUDE** angeben.

Die .MAK-Datei ist nur eine Textdatei. Wenn Sie diese bearbeiten möchten, laden Sie die Datei in QuickBASIC, indem Sie die Option "Dokument" des Befehls **Datei laden** aus dem Menü **Datei** benutzen.

4.4.2 Wie Sie Module aus einem Programm entfernen (Datei entfernen)

Wählen Sie den Befehl **Datei entfernen** aus dem Menü **Datei**, um ein gesamtes Modul aus Ihrem Programm zu entfernen. Nachdem Sie ein Modul entfernt haben, existiert die Datei, die das Modul enthält, noch auf Diskette/Festplatte; aber wenn Sie Ihr Programm speichern, ist das Modul nicht länger Teil des Programms, und der Name des Moduls erscheint nicht in der .MAK-Datei des Programms.

Wenn Sie unabsichtlich ein Modul laden und sich anschließend entscheiden, es nicht in Ihrem Programm zu verwenden, folgen Sie diesen Schritten, um es zu entfernen:

1. Wählen Sie den Befehl **Datei entfernen** aus dem Menü **Datei**.
2. Heben Sie den Namen des Moduls, das Sie nicht mehr in Ihrem Programm verwenden möchten, hervor.
3. Betätigen Sie die EINGABETASTE. Das Modul ist entfernt, existiert jedoch noch als Diskettendatei.

4.5 Include-Dateien

Include-Dateien sind Textdateien, die mit einem **$INCLUDE**-Metabefehl in ein Programm kompiliert werden. Sie sind Modulen darin ähnlich, daß Sie dieselben Include-Dateien in vielen verschiedenen Programmen verwenden können. Anders als Module dürfen Include-Dateien jedoch keine **SUB-**, **END SUB-**, **FUNCTION-** oder **END FUNCTION**-Anweisungen enthalten.

Include-Dateien werden stets im Textformat gespeichert, selbst wenn Sie die Option "QuickBASIC - Schnell laden und speichern" aus dem Dialogfeld des Befehls **Speichern unter** gewählt haben.

Hinweis Wenn Sie QuickBASIC 2.0- oder 3.0-Programme haben, die zur Definition von **SUB**-Prozeduren Include-Dateien verwenden, finden Sie in Kapitel 4.7, "Die Inhalte zweier Dateien mischen", eine Erklärung, wie diese Prozeduren in Ihr QuickBASIC Version 4.0-Programm eingebunden werden können.

Obwohl Sie Include-Dateien auf jede beliebige Weise verwenden können, sind sie besonders hilfreich bei der Zusammenfassung deklarierender Anweisungen, die Auswirkungen auf mehrere Module eines mehrmoduligen Programms haben. Beispiele deklarierender Anweisungen sind die folgenden:

Anweisung	*Verwendung*
COMMON	Führt Variablen auf, die mehr als einem Modul gemeinsam sind
DECLARE	Deklariert **SUB**- oder **FUNCTION**-Prozeduren und ermöglicht Typüberprüfung für deren Argumente
DIM	Deklariert Variablentypen
TYPE	Deklariert benutzerdefinierte Typen

Weitere Informationen zu diesen Anweisungen finden Sie im *BASIC-Befehlsverzeichnis* und in Kapitel 2, "Prozeduren: Unterprogramme und Funktionen", in *Programmieren in BASIC: Ausgewählte Themen.*

Innerhalb Ihrer Programme können Sie **$INCLUDE**-Metabefehle an beliebiger Stelle plazieren. Wenn die Include-Datei jedoch deklarierende Anweisungen enthält, wird der Metabefehl **$INCLUDE** normalerweise so nah wie möglich am Beginn des Moduls plaziert, damit der Geltungsbereich der Deklarationen das gesamte Modul umfaßt und weil einige deklarierende Anweisungen vor ausführbaren Anweisungen stehen müssen.

4.5.1 Wie Sie Include-Dateien erstellen und laden (Datei anlegen)

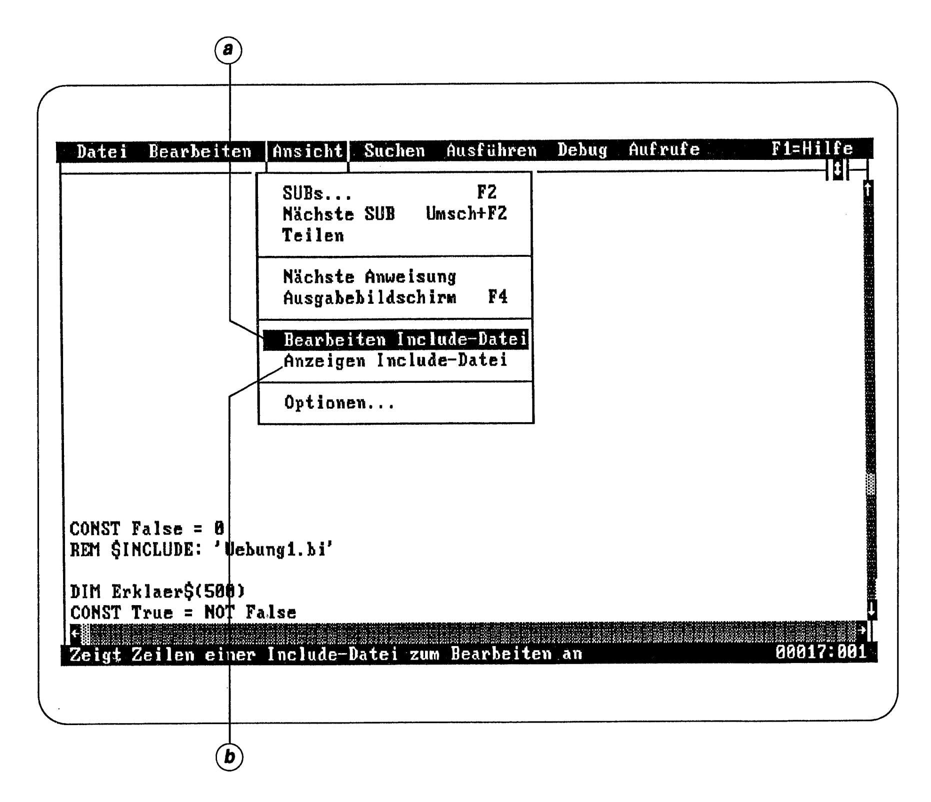
Wenn Sie eine Include-Datei anlegen möchten, wählen Sie den Befehl **Datei anlegen** aus dem Menü **Datei** und wählen anschließend die Option "Include", um die Datei als Include-Datei zu klassifizieren.

Der wichtigste Punkt, an den Sie sich beim Anlegen oder Laden irgendeiner Datei erinnern müssen, besteht darin, daß die Datei in dem Optionsfeld "Anlegen als" richtig klassifiziert wird. Wenn Sie zum Beispiel eine Include-Datei laden oder anlegen und nicht ausdrücklich die Option "Include" markieren, lädt QuickBASIC die Datei als ein Programmodul. Wenn Sie Ihr Programm dann speichern, schreibt QuickBASIC den Namen der Include-Datei zusammen mit den Namen Ihrer anderen Programmodule in die .MAK-Datei. Beim nächsten Laden des Programms wird die Include-Datei mit den anderen Moduldateien geladen sowie zusätzlich als eine Include-Datei kompiliert. Die Plazierung eines Include-Dateinamens in Ihrer .MAK-Datei macht keinen Sinn und verschwendet Speicherplatz.

4.5.2 Wie Sie sich Include-Dateien ansehen und diese bearbeiten (Bearbeiten Include-Datei, Anzeigen Include-Datei)

Benutzen Sie die Befehle **Bearbeiten Include-Datei** und **Anzeigen Include-Datei** aus dem Menü **Ansicht**, um diese zu bearbeiten und Include-Dateien anzusehen (siehe Abbildung 4.8).

*Abbildung 4.8 Die Include-Datei-Befehle aus dem Menü **Ansicht***



*a) Wählen Sie **Bearbeiten Include-Datei**, um eine Include-Datei an der Cursorposition zu bearbeiten.*

*b) Wählen Sie **Anzeigen Include-Datei**, um den Inhalt einer Include-Datei nur zur Ansicht anzuzeigen.*

Wenn Sie sich eine Include-Datei ansehen, diese aber nicht bearbeiten möchten, dann wählen Sie den Befehl **Anzeigen Include-Datei**. Dieser Befehl ist ein "Umschalter"; das heißt, **Anzeigen Include-Datei** schaltet diese Option ein, wenn sie ausgeschaltet ist, und aus, wenn sie bereits eingeschaltet ist. Ein Prüfzeichen (√) erscheint direkt neben der Option, wenn sie eingeschaltet ist. Dies ermöglicht es Ihnen, den Inhalt der Include-Datei zu betrachten, der unter dem entsprechenden **$INCLUDE**-Metabefehl mit hoher Intensität erscheint. Obwohl Sie sich durch die Datei bewegen können, nachdem Sie **Anzeigen Include-Datei** gewählt haben, können Sie die Datei solange nicht bearbeiten, wie diese Option eingeschaltet ist. Falls Sie versuchen, eine der eingefügten Zeilen zu bearbeiten, während diese hervorgehoben sind, erscheint ein Dialogfeld, das Sie fragt, ob Sie die Include-Datei bearbeiten möchten. Wenn Sie dies wünschen, lädt QuickBASIC die Datei in den Speicher und plaziert deren Text im aktiven Arbeitsbereich.

Sollten Sie versuchen, andere Zeilen als solche in der Include-Datei zu bearbeiten, erscheint ein Dialogfeld, das Sie fragt, ob Sie die Option **Anzeigen Include-Datei** ausschalten möchten. Wenn Sie die EINGABETASTE betätigen, verschwinden die hervorgehobenen Zeilen der Include-Datei.

Wenn Sie sich die Include-Datei ansehen und diese bearbeiten möchten, wählen Sie den Befehl **Bearbeiten Include-Datei**. Falls sich der Cursor auf einer Zeile mit einem **$INCLUDE**-Metabefehl befindet, bewegt dieser Befehl den Text der Include-Datei zum Bearbeiten in das aktive Fenster. Beachten Sie, daß QuickBASIC den eigentlichen Text von Include-Dateien getrennt vom restlichen Programmtext hält. Dies ist ähnlich der Art, in der die Texte von **SUB**- oder **FUNCTION**-Prozeduren von anderen Programmtexten getrennt sind. Der Unterschied besteht darin, daß QuickBASIC den Text einer Include-Datei nicht als untergeordnet zu dem übrigen Programm betrachtet, wie es dies bei Prozeduren tut.

Nachdem eine Include-Datei einmal existiert, möchten Sie diese vielleicht bearbeiten, ohne zu der Stelle Ihres Programmes zurückzukehren, die den **$INCLUDE**-Metabefehl enthält, der sich auf die Include-Datei bezieht. Um dies durchzuführen

1. Teilen Sie Ihren Bildschirm mit dem Befehl **Teilen** aus dem Menü **Ansicht**.
2. Wählen Sie den Befehl **Datei laden** aus dem Menü **Datei**, um die Include-Datei zu laden. Vergessen Sie nicht, die Option "Include" zu markieren.
3. Bearbeiten Sie die Include-Datei.
4. Wählen Sie den Befehl **Speichern** aus dem Menü **Datei**, um alle Änderungen, die Sie an der Include-Datei vorgenommen haben, abzuspeichern.
5. Betätigen Sie F6, um zu dem Programmteil im anderen Arbeitsbereich zurückzugelangen.
6. Wählen Sie den Befehl **Teilen** aus dem Menü **Ansicht** erneut, um das zusätzliche Fenster zu schließen.

Bevor bei der nächsten Ausführung des Programmes Änderungen berücksichtigt werden können, müssen Sie die Include-Datei abspeichern. QuickBASIC fordert Sie dazu auf, falls Sie es vergessen.

4.5.3 Include-Dateien verschachteln

Es gibt keine obere Grenze für die Anzahl an **$INCLUDE**-Metabefehlen, die Sie in den Modulen eines Programmes haben können, es gibt aber eine obere Grenze für das Ausmaß der "Verschachtelungen".

Verschachtelungen treten auf, wenn ein **$INCLUDE**-Metabefehl in einer weiteren Include-Datei erscheint. Include-Dateien können auf bis zu fünf Ebenen verschachtelt werden. Falls Sie in einer Include-Datei einen kreisförmigen Bezug haben (das heißt, einen **$INCLUDE**-Metabefehl, der auf sich selbst oder eine Datei, in der auf ihn Bezug genommen wird, Bezug nimmt), erhalten Sie die Fehlermeldung `Zu viele Dateien`, wenn Sie versuchen, Ihr Programm zu starten.

Hinweis Wenn QuickBASIC einen **$INCLUDE**-Metabefehl verarbeitet, wird nur das aktuelle Arbeitsverzeichnis (oder ein in dem Metabefehl selbst angegebenes Verzeichnis) durchsucht. Falls Sie Ihre Include-Dateien alle in einem zentralen Verzeichnis zusammenfassen möchten, können Sie diese für QuickBASIC erreichbar werden lassen, indem Sie mit einem vollen oder relativen Pfadnamen durchgehend auf sie Bezug nehmen. Zum Beispiel weist die folgende Zeile QuickBASIC an, nach der DEKLA.BI benannten Include-Datei in dem Verzeichnis C:\INCLUDE zu suchen:

```
' $INCLUDE: 'c:\include\dekla.bi'
```

Dies kann auch mit einem relativen Pfad angegeben werden, wie in der folgenden Zeile:

```
' $INCLUDE: '..\include\dekla.bi'
```

Obwohl jede Erweiterung, einschließlich .BAS, für Dateien, die mit dem Metabefehl **$INCLUDE** in Ihr Programm eingebunden werden, zulässig ist, wird empfohlen, daß Include-Dateien nur eine einzige Erweiterung (zum Beispiel .BI) bekommen, um Verwechslungen mit Programmodulen zu vermeiden.

4.6 Dokumente

Wenn Sie eine Datei als Dokument anlegen oder laden, schalten Sie die Eigenschaften des intelligenten QuickBASIC-Editors aus; das heißt, QuickBASIC formatiert weder den von Ihnen eingegebenen Text, noch überprüft es die Syntax, um sicherzustellen, daß Sie gültige BASIC-Anweisungen eingeben.

Wenn Sie QuickBASIC zur Editierung einer Datei einsetzen möchten, die keinen BASIC-Code enthält, öffnen Sie das Menü **Datei** und wählen anschließend den Befehl **Datei anlegen**, um ein neues Dokument zu erstellen, oder den Befehl **Datei laden**, um ein existierendes Dokument zu bearbeiten. Vergessen Sie nicht, die Datei als ein Dokument anzugeben, indem Sie in dem entsprechend folgenden Dialogfeld die Option "Dokument" markieren.

Eine als Dokument geladene Datei kann unter anderen Umständen ein Programmodul darstellen – das heißt, sie kann gültigen Quellcode in der Sprache BASIC enthalten. Wenn Sie jedoch die Datei als ein Dokument in QuickBASIC laden, weisen Sie QuickBASIC an, diese nicht als Programmbestandteil zu behandeln. Konsequenterweise können Sie ein Programm nicht starten, wenn es als Dokument geladen ist. Eine als Dokument geladene Datei kann nicht mit der Option "QuickBASIC – Schnell laden und speichern" des Befehls **Speichern unter** abgespeichert werden. Sie können diese Option zwar wählen, QuickBASIC speichert ein Dokument aber immer im Textformat ab. Umgekehrt erhalten Sie, wenn Sie ein Programm oder Modul mit der Option "QuickBASIC – Schnell laden und speichern" abgespeichert haben, die Fehlermeldung `Falscher Dateimodus`, wenn Sie versuchen, die Datei als ein Dokument zu laden. Um eine solche Datei als ein Dokument zu laden, müssen Sie sie zunächst mit der Option "Text – Von anderen Programmen lesbar" abspeichern.

Um ein Programm zu starten, nachdem Sie es als ein Dokument geladen haben, wählen Sie den Befehl **Programm laden** aus dem Menü **Datei**.

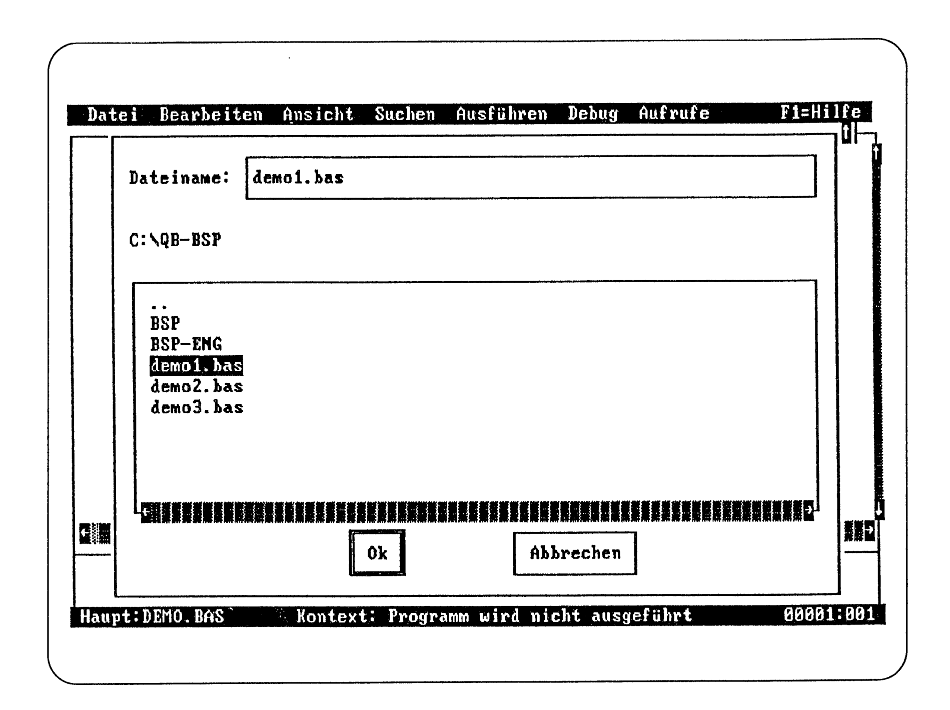
Wenn die Dialogfelder **Datei laden** bzw. **Datei anlegen** auf dem Bildschirm erscheinen, ist die Option "Modul" bereits markiert. Sie müssen daher, wenn Sie eine Datei als Include-Datei oder Dokument laden möchten, ausdrücklich die Option "Include" oder "Dokument" markieren. Wenn Sie das erstemal mit QuickBASIC arbeiten, kann es vorkommen, daß Sie Dateien aus Versehen als Module laden. Wenn Sie der Meinung sind, daß dies passiert ist, markieren Sie den Namen der Datei im Dialogfeld **SUBs** aus dem Menü **Ansicht** und beachten die Beschreibung unter dem Listenfeld. Falls die Datei mit einer Klassifikation versehen ist, die sich von derjenigen unterscheidet, die Sie erwarten, wählen Sie die Befehlsfläche "Löschen", um die Datei aus dem Speicher zu entfernen. Dies hat keine Auswirkungen auf die Datei auf Diskette, erlaubt es Ihnen aber, die Datei nochmals richtig zu laden.

Hinweis Solange Sie noch nicht so sehr vertraut mit der QuickBASIC-Umgebung sind, sollten Sie beim Löschen von Namen aus dem Listenfeld des Dialogfeldes **SUBs** vorsichtig sein. Erinnern Sie sich: Während Module unabhängig als Diskettendateien existieren, existieren **SUB**- und **FUNCTION**-Prozeduren nur innerhalb von Modulen. Wenn Sie eine Prozedur löschen, entfernen Sie diese nicht zeitweise aus dem Speicher; stattdessen löschen Sie die Prozedur komplett aus der Diskettendatei. Daher ist diese Prozedur für immer verschwunden, wenn Sie ein Programm nach dem Löschen eines Prozedurnamens aus dem Listenfeld abspeichern.

4.7 Die Inhalte zweier Dateien mischen (Zusammenführen)

Der Befehl **Zusammenführen** aus dem Menü **Datei** fügt im aktiven Fenster eine Datei vor der Cursor-Position ein. Wenn Sie den Befehl **Zusammenführen** aus dem Menü **Datei** wählen, öffnet sich das in Abbildung 4.9 gezeigte Dialogfeld:

Abbildung 4.9 Dialogfeld ***Zusammenführen***



Das Dialogfeld **Zusammenführen** arbeitet in derselben Weise wie das Dialogfeld **Programm laden**. Der Befehl **Zusammenführen** fügt die Datei jedoch in das aktuelle Modul ein, während der Befehl **Programm laden** alles löscht, was sich aktuell im Speicher befindet, und anschließend das Programm lädt. Erläuterungen zur Verwendung des Dialogfeldes **Programm laden** sowie zur Auflistung des Inhalts des aktuellen Verzeichnisses finden Sie in den Abschnitten 4.2.1.1 bis 4.2.1.2.

QuickBASIC, Version 4.0, erlaubt keine **SUB-** oder **FUNCTION**-Prozeduren in Include-Dateien. Wenn Sie in QuickBASIC 2.0 oder 3.0 erstellte Programme haben, die auf Unterprogramme in Include-Dateien angewiesen sind, verwenden Sie den Befehl **Zusammenführen**, um die Include-Datei mit Ihrem Hauptmodul zu mischen. (Vergessen Sie nicht, auch den Metabefehl **$INCLUDE** zu löschen.)

Wenn Sie eine Include-Datei mischen, die eine **SUB**-Prozedur enthält, erscheint der Text der Prozedur nicht in dem aktuell aktiven Fenster. Sobald Sie den Befehl **Zusammenführen** ausführen, können Sie Ihr Programm jedoch starten.

Um die eingemischte **SUB**-Prozedur zu betrachten oder zu bearbeiten, wählen Sie **SUBs** aus dem Menü **Ansicht** und anschließend den Namen der eingemischten Prozedur aus dem Listenfeld.

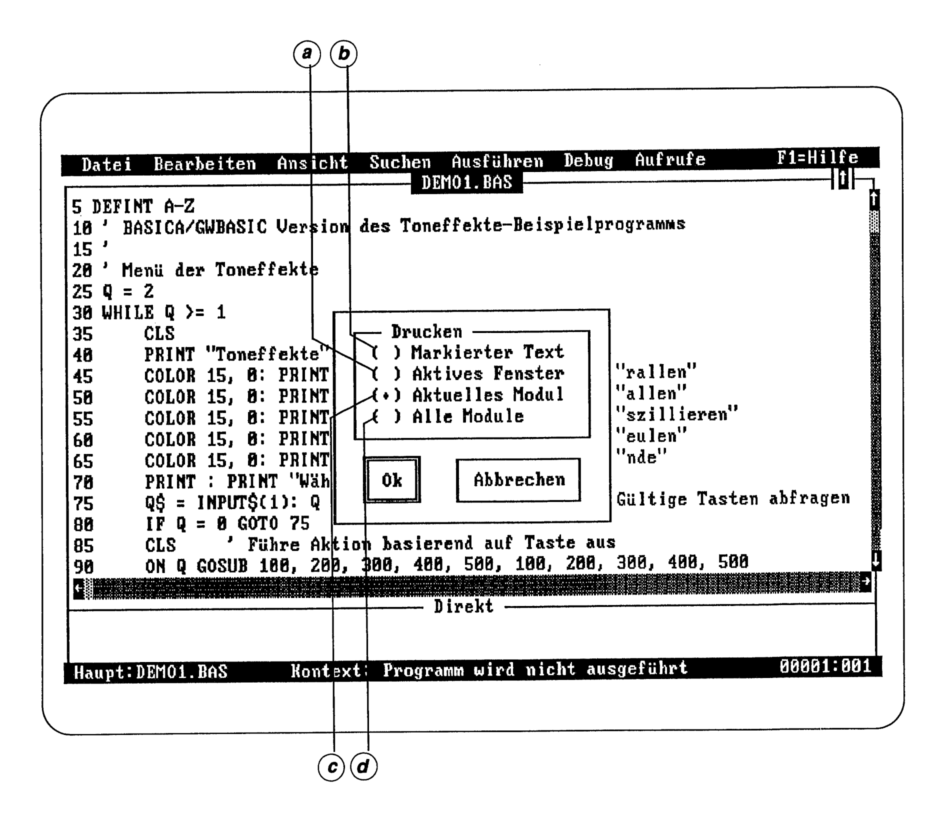
Anstatt sie einzumischen, möchten Sie Ihre **SUB**-Prozedur vielleicht umschreiben und sie in ein separates Modul geben. In diesem Fall können einige Umstrukturierungen notwendig werden, falls die **SUB** mit dem Attribut **SHARED** deklarierte Variablen verwendet. Weitere Informationen finden Sie in Kapitel 2, "Prozeduren: Unterprogramme und Funktionen", in *Programmieren in BASIC: Ausgewählte Themen* sowie in Kapitel 4, "Programme und Module", im *BASIC-Befehlsverzeichnis.*

4.8 Wie Sie Dateien drucken (Drucken)

Der Befehl **Drucken** aus dem Menü **Datei** ermöglicht es Ihnen, markierten Text oder Text des aktiven Fensters, des aktuellen Moduls oder des aktuellen Programms zu drucken. Um den Befehl **Drucken** verwenden zu können, muß Ihr Drucker mit LPT1 verbunden sein.

Wenn Sie den Befehl **Drucken** aus dem Menü **Datei** wählen, öffnet sich das in Abbildung 4.10 gezeigte Dialogfeld:

*Abbildung 4.10 Das Dialogfeld **Drucken***



a) *Druckt den Inhalt des aktiven Fensters, entweder Modul-Ebenen-Code oder eine **SUB**- oder **FUNCTION**-Prozedur.*

b) *Druckt nur den im aktiven Fenster markierten Text.*

c) *Druckt das gesamte mit dem aktiven Fenster verknüpfte Modul.*

d) *Druckt alle Module, Include-Dateien und Dokument-Dateien, die sich aktuell im Speicher befinden.*

Wählen Sie die passende Option aus dem Optionsfeld **Drucken**, um Ihre Datei zu drucken.

5 Bearbeiten

Dieses Kapitel beschreibt, wie mit den QuickBASIC-Editierbefehlen sowie den Menüs **Bearbeiten** und **Suchen** ein Programmtext eingegeben und bearbeitet wird. Bevor Sie mit diesem Kapitel beginnen, sollten Sie mit dem QuickBASIC-Bildschirm und den in Kapitel 3, "Wie Sie sich in der QuickBASIC-Umgebung zurechtfinden", beschriebenen Techniken vertraut sein.

Nachdem Sie dieses Kapitel gelesen haben, werden Sie wissen, wie

- Einfache Editierfunktionen, wie zum Beispiel Text eingeben und den Cursor bewegen, durchgeführt werden.
- Text gelöscht bzw. eingefügt wird.
- Textblöcke verschoben oder kopiert werden.
- Nach Zeichen, Wörtern oder einer Gruppe von Wörtern gesucht wird, bzw. diese ersetzt werden.
- Text von anderen Dateien kopiert wird.

5.1 Wie Sie Text eingeben

Wenn Sie ein BASIC-Programm eintippen, können Sie eine Zeile abschließen entweder durch Betätigung der EINGABETASTE am Ende der Zeile, oder indem Sie den Cursor mit einer RICHTUNGSTASTE oder dem Mauszeiger von der Zeile wegbewegen.

5.2 Die Eigenschaften des intelligenten Editors

Die meisten Compiler zwingen Sie dazu, Ihren Quellcode mit einem Texteditor zu schreiben und die daraus resultierende Textdatei anschließend zu kompilieren. QuickBASIC hat jedoch schon immer sowohl einen Texteditor als auch einen Compiler in einer integrierten Umgebung geboten. QuickBASIC, Version 4.0, geht einen Schritt weiter und kombiniert einen Texteditor, eine besondere Art von Compiler sowie einen Debugger zu einem einzigen intelligenten Editor. Der intelligente Editor umfaßt besondere Eigenschaften, wie zum Beispiel automatische Syntaxüberprüfung, die es erleichtern, BASIC-Programme einzugeben und zu bearbeiten. Wenn der intelligente Editor eingeschaltet ist, führt QuickBASIC jedesmal, wenn Sie eine Zeile eingeben, die folgenden Aktionen durch:

- Überprüfen der Zeile auf Syntaxfehler.
- Formatieren der Zeile wie benötigt.
- Übersetzen der Zeile in ausführbare Form, sofern die Syntax korrekt ist. Das bedeutet, daß jede von Ihnen eingegebene Zeile vorbereitet ist, sofort ausgeführt zu werden.

5.2.1 Wann ist der intelligente Editor eingeschaltet?

In den meisten Fällen verwenden Sie den QuickBASIC-Editor, um Programmtext einzugeben und zu bearbeiten. Wenn Sie QuickBASIC starten und einfach mit dem Eintippen beginnen, geht QuickBASIC davon aus, daß Sie BASIC-Anweisungen schreiben möchten, so daß die Eigenschaften des intelligenten Editors automatisch eingeschaltet werden. Der intelligente Editor ist ebenfalls eingeschaltet, wenn Sie

- **Neues Programm** oder **Programm laden** aus dem Menü **Datei** wählen.
- QuickBASIC mit dem Basisnamen eines Programmes, das Sie bearbeiten möchten, starten.
- **Datei anlegen** oder **Datei laden** aus dem Menü **Datei** und anschließend die Option "Modul" oder "Include" wählen.

Gelegentlich müssen Sie vielleicht eine Textdatei bearbeiten, die etwas anderes als BASIC-Anweisungen enthält; solche Dateien umfassen die .MAK-Datei von QuickBASIC sowie Dokument-Dateien, die Ihr Programm eventuell benötigt. Für diesen Zweck können Sie die Eigenschaften des intelligenten Editors ausschalten und den QuickBASIC-Editor als normales Text-System einsetzen. Dies kann erreicht werden durch die Wahl von **Datei anlegen** oder **Datei laden** aus dem Menü **Datei** und anschließende Wahl der Option "Dokument".

5.2.2 Automatische Syntaxüberprüfung

Wenn der intelligente Editor eingeschaltet ist, überprüft er die Syntax jeder Zeile, wenn Sie diese abschließen. Ein Syntaxfehler zeigt an, daß die Zeile irgendetwas enthält, das keine zulässige QuickBASIC-Anweisung darstellt. Zum Beispiel erzeugt diese Zeile einen Syntaxfehler, da das Schlüsselwort **GOTO** kein gültiges Argument für die Anweisung **PRINT** ist:

```
PRINT GOTO
```

Wenn ein Fehler auftritt, zeigt QuickBASIC ein Dialogfeld an, das eine Fehlermeldung enthält. Die Betätigung der ESC-TASTE oder der LEERTASTE löscht die Meldung vom Bildschirm und positioniert den Cursor dort, wo der Fehler auftrat. Bevor Ihr Programm starten kann, müssen Sie Syntaxfehler korrigieren. Falls Sie ohne Korrektur des Fehlers fortfahren, erscheint die Meldung erneut, nachdem Sie das Programm gestartet haben.

Hinweis Wenn Sie eine Syntaxfehlermeldung löschen, wird der Cursor auf der Anweisung plaziert, die den Fehler verursacht hat. Sie können den Cursor auf einen beliebigen Teil des BASIC-Schlüsselwortes bewegen und anschließend UMSCHALTTASTE+F1 betätigen, um direkte Hilfe zur Syntax der Anweisung zu bekommen.

Bestimmte Tippfehler werden bei der Eingabe nicht als Syntaxfehler erkannt. Zum Beispiel erscheint keine Fehlermeldung, wenn Sie das Schlüsselwort **PRINT** bei der Eingabe dieser Zeile falsch eintippen:

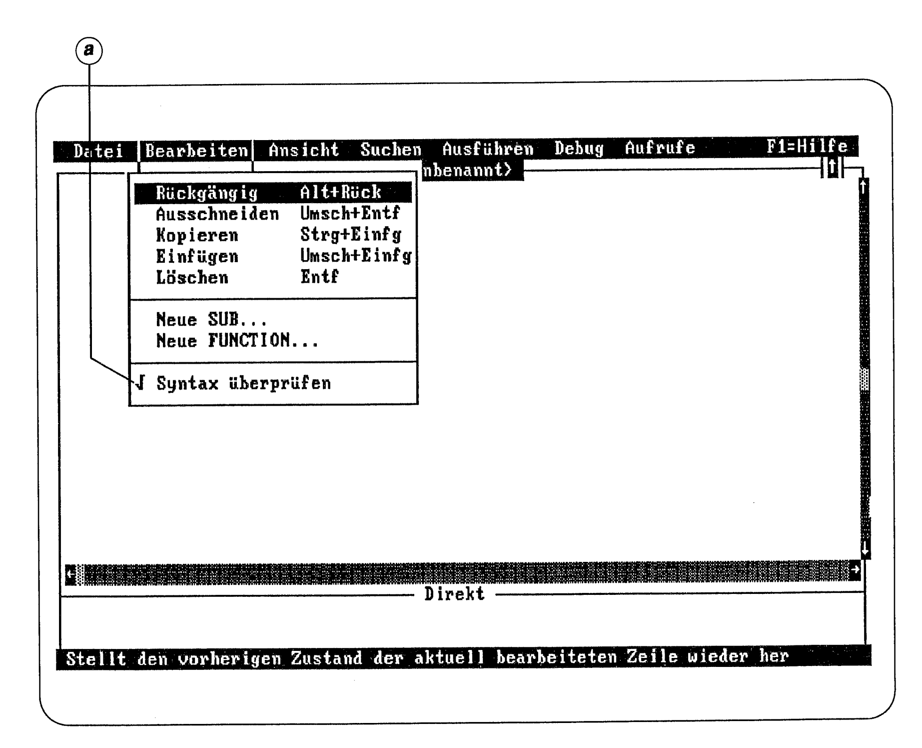
```
pront
```

Obwohl dies nicht wie eine gültige BASIC-Anweisung aussieht, akzeptiert QuickBASIC diese Zeile solange, bis Sie versuchen, das Programm zu starten. Zu diesem Zeitpunkt wird der Fehler signalisiert. Bis dahin wird die Anweisung als Aufruf einer als "`pront`" benannten **SUB**-Prozedur interpretiert.

5.2.3 Wie Sie die Syntaxüberprüfung ausschalten

Die Syntaxüberprüfung ist immer eingeschaltet, solange Sie nicht, wie im vorhergehenden Abschnitt erläutert, eine Datei mit der Option "Dokument" geladen haben. Sie können die Syntaxüberprüfung ein- und ausschalten mit dem Befehl **Syntax überprüfen** aus dem in Abbildung 5.1 gezeigten Menü **Bearbeiten**. Ein Prüfzeichen erscheint direkt neben dem Befehl, wenn die Syntaxüberprüfung eingeschaltet ist.

Abbildung 5.1 Eingeschaltete Syntaxüberprüfung im Menü ***Bearbeiten***



a) Die Syntaxüberprüfung ist eingeschaltet.

5.2.4 Automatische Formatierung

Zusätzlich zur Überprüfung auf Syntaxfehler standardisiert der intelligente Editor automatisch das Format jeder Zeile, während Sie diese einfügen. Nach der Formatierung

- Erscheinen alle QuickBASIC-Schlüsselwörter in Großbuchstaben.
- Erscheinen Leerzeichen vor und hinter allen BASIC-Operatoren.
- Sind die Namen von Variablen und Prozeduren einheitlich mit großen Anfangsbuchstaben geschrieben.
- Sind Satzzeichen passend eingefügt; zum Beispiel fügt der intelligente Editor Semikolons zwischen den einzelnen Komponenten einer **PRINT**-Anweisung ein, wenn Sie diese nicht selbst einfügen, und ergänzt ein passendes abschließendes Anführungszeichen hinter dem Argument einer **PRINT**-Anweisung, wenn Sie dies vergessen haben.

QuickBASIC merkt sich die Namen von allen Variablen und Prozeduren Ihres Programms und schreibt diese Namen einheitlich mit großen Anfangsbuchstaben. Nehmen Sie zum Beispiel an, Sie haben eine als `meinvar` benannte Variable, die Sie zunächst nur mit kleingeschriebenen Buchstaben eingeben. Wenn Sie sich später entschließen, den Namen in `MeinVar` zu ändern, wobei sowohl `M` als auch `V` groß geschrieben werden, ändert QuickBASIC sämtliche in dem Modul vorkommenden Namen `meinvar` in `MeinVar`. Wenn Sie das Programm speichern oder starten und anschließend in den Editor zurückkehren, können Sie sehen, daß der Name `meinvar` in `MeinVar` geändert ist.

5.3 Einfügen und Überschreiben

Neue einzugebende Zeichen werden dem Text entweder durch Einfügen oder Überschreiben hinzugefügt. Wenn der Schalter "Einfügen" gesetzt ist, so daß der Cursor eine einzige blinkende Linie bildet, fügt der Editor jedes neue Zeichen an der Cursor-Position ein. Wenn Sie den Schalter Einfügen einsetzen, um den Cursor in ein blinkendes Viereck zu verwandeln, überschreibt jedes neue Zeichen das Zeichen unter dem Cursor und ersetzt es. Sowohl die EINFG-TASTE als auch STRG+V schalten die Form des Cursors um.

5.4 Wie Sie Text markieren

Obwohl einfache Editierwerkzeuge, wie zum Beispiel die RÜCKTASTE und die RICHTUNGSTASTEN, für kleine Editieraufgaben völlig ausreichend sind, ist es sehr häufig vorteilhaft, mit Text in größeren Blöcken zu arbeiten. Bevor Sie eine umfangreiche Editieroperation, wie zum Beispiel kopieren oder löschen eines Textblockes, durchführen können, müssen Sie den Textausschnitt, den Sie bearbeiten möchten, anzeigen, indem Sie ihn markieren (hervorheben). Halten Sie die UMSCHALTTASTE gedrückt, während Sie eine RICHTUNGSTASTE betätigen, um Text zu markieren. Zum Beispiel markiert UMSCHALTTASTE+NACH RECHTS (→) Text, der rechts vom Cursor steht.

Das Halten der UMSCHALTTASTE während der Verwendung einer der Tastenkurzkombinationen für die Editierung markiert Text ebenso. Zum Beispiel wird durch Betätigen von UMSCHALTTASTE+ENDE alles vom Cursor an bis zum Zeilenende markiert. Um den Text wieder in seinen unmarkierten Zustand zurückzuführen, müssen Sie eine beliebige Richtungstaste betätigen.

Hinweis Wenn Sie Text markieren, wird der Cursor am Ende des markierten Textes plaziert. Wenn Sie mit dem Eintippen beginnen, während der Text noch markiert ist, ersetzen Ihre Eingaben den markierten Text.

Wenn Sie Text von einer Stelle zu einer anderen kopieren, bleibt der Originaltext markiert. Wenn Sie diesen nicht in seinen unmarkierten Status zurücksetzen, ersetzt alles, was Sie als nächstes eintippen (während sich der markierte Text im aktiven Arbeitsbereich befindet), den markierten Text.

5.5 Text löschen und einfügen

Sie können mit den Befehlen des Menüs **Bearbeiten** Text löschen und einfügen. Abschnitt 5.5.1 erläutert die Zwischenablage, einen temporären Speicherbereich für gelöschten Text. Abschnitt 5.5.2 erklärt die Befehle zum Löschen und Einfügen von Text.

5.5.1 Wie Sie die Zwischenablage einsetzen

Die Zwischenablage ist ein temporärer Speicherbereich, der Text enthält, den Sie aus dem aktiven Fenster kopieren oder löschen. Text wird in der Zwischenablage nicht gesammelt. Alles, was Sie in der Zwischenablage plazieren, ersetzt alles, was sich dort bereits befindet.

Text verbleibt solange in der Zwischenablage, bis Sie Ihre QuickBASIC-Übung beenden oder den Text durch neuen Text ersetzen. (Die Befehle **Neues Programm** und **Programm laden** löschen die Zwischenablage nicht.) Wenn die Zwischenablage einmal Text enthält, können Sie diesen Text, sooft Sie es wünschen, einmischen oder einfügen.

Hinweis Ausschneiden oder Kopieren eines großen Textblockes nimmt viel Speicherplatz in Anspruch. Wenn Sie die Zwischenablage für große Textblöcke verwenden, sollten Sie diese anschließend leeren, indem Sie einen kleinen Textblock hineinkopieren, wie zum Beispiel ein einzelnes Zeichen.

5.5.2 Die Befehle Löschen, Ausschneiden, Kopieren und Einfügen

Das Menü **Bearbeiten** enthält vier Befehle – **Löschen, Ausschneiden, Kopieren** und **Einfügen** – die es Ihnen ermöglichen, Textblöcke einzufügen und zu löschen:

Befehl und Tastenkurzkombination	*Funktion*
Löschen (ENTF-TASTE)	Löscht markierten Text komplett
Ausschneiden (UMSCHALTTASTE+ENTF)	Löscht einen markierten Text und plaziert ihn in der Zwischenablage
Kopieren (STRG+EINFG-TASTE)	Plaziert eine Kopie des markierten Textes in der Zwischenablage, beläßt den Originaltext unverändert
Einfügen (UMSCHALTTASTE+EINFG)	Fügt Text aus der Zwischenablage an der Cursor-Position ein

Um **Löschen, Ausschneiden** oder **Kopieren** zu verwenden, müssen Sie den Text, den Sie löschen oder kopieren möchten, zunächst markieren und anschließend **Löschen, Ausschneiden** oder **Kopieren** aus dem Menü **Bearbeiten** wählen. Der Befehl **Einfügen** fügt Text aus der Zwischenablage in den aktuellen Text ein. Dieser Befehl funktioniert nur, wenn die Zwischenablage Text enthält, und das Ergebnis des Befehls hängt davon ab, ob gleichzeitig irgendein Text markiert ist. Beachten Sie, daß **Einfügen** den markierten Text durch den Inhalt der Zwischenablage *ersetzt*, falls Sie im aktiven Fenster markierten Text vorliegen haben. Wenn kein Text markiert ist, fügt **Einfügen** den Inhalt der Zwischenablage links vom Cursor ein, oder, wenn der Block eine Zeile überschreitet, in der Zeile oberhalb des Cursors.

Verwenden Sie diese Schritte, um Text einzufügen:

1. Bewegen Sie den Cursor an die Stelle, an der Sie den Text einfügen möchten. Der neue Text wird links des Cursors eingefügt, oder, wenn der Block eine Zeile überschreitet, in der Zeile oberhalb des Cursors.
2. Wählen Sie den Befehl **Einfügen** aus dem Menü **Bearbeiten** (oder betätigen Sie UMSCHALTTASTE+EINFG).

5.6 Wie Sie die letzte Editierung rückgängig machen

Der Befehl **Rückgängig** aus dem Menü **Bearbeiten** (ALT+RÜCKTASTE) führt die aktuelle Zeile in ihren Originalzustand zurück, indem alle Änderungen zurückgenommen werden. Sie können eine Zeile solange wiederherstellen, wie der Cursor sich auf ihr befindet. Wenn Sie den Cursor einmal von der Zeile wegbewegt haben, sind alle von Ihnen durchgeführten Änderungen dauerhaft.

Rückgängig funktioniert nur, wenn Sie gerade an der aktuellen Zeile eine Änderung vorgenommen haben. Wenn Sie versuchen, **Rückgängig** zu wählen, während sich der Cursor auf einer nicht veränderten Zeile befindet, geschieht nichts.

Hinweis Sie können **Rückgängig** nicht verwenden, um eine mit STRG+Y gelöschte Zeile wiederherzustellen, da STRG+Y die gesamte Zeile entfernt. Sie können die Zeile aber mit dem Befehl **Einfügen** wiederherstellen, da eine mit STRG+Y gelöschte Zeile in der Zwischenablage gespeichert wird.

5.7 Wie Sie Text bewegen und kopieren

Sie können Textblöcke mit den Befehlen **Ausschneiden**, **Kopieren** und **Einfügen** aus dem Menü **Bearbeiten** bewegen oder kopieren. Das Bewegen von Text plaziert eine Kopie des markierten Textes an der neuen Position und löscht den Originaltextblock, während das Kopieren von Text eine Kopie des markierten Textes an einer neuen Position plaziert, ohne das Original zu zerstören.

Verwenden Sie diese Schritte, um einen Textblock zu bewegen:

1. Markieren Sie den Text, den Sie bewegen möchten.
2. Wählen Sie den Befehl **Ausschneiden** aus dem Menü **Bearbeiten** (oder betätigen Sie UMSCHALTTASTE+ENTF), um den Text zu löschen. Der markierte Text wird in die Zwischenablage kopiert und ersetzt alles, was sich dort befindet.
3. Bewegen Sie den Cursor, abhängig von der Zielposition für den Text, wie folgt auf die neue Position:
   - Um den Text in dem aktuell aktiven Fenster zu plazieren, verwenden Sie die normalen Befehle zur Cursorbewegung, um sich zu der neuen Position zu bewegen. Andernfalls verwenden Sie F6, um das Fenster zu aktivieren, in das Sie den Textblock bewegen möchten.
   - Wenn Sie den Text in einer **SUB**- oder **FUNCTION**-Prozedur plazieren möchten, wählen Sie zunächst den Befehl **SUBs** aus dem Menü **Ansicht** und anschließend die Zielprozedur. Wenn die Prozedur erscheint, bewegen Sie den Cursor auf die neue Position in dieser Prozedur, indem Sie die Befehle zur Cursorbewegung verwenden.
   - Wenn Sie den Text in einem Programm oder Modul, das sich aktuell nicht im Speicher befindet, plazieren möchten, wählen Sie zunächst den Befehl **Neues Programm**, **Programm laden**, **Datei laden** oder **Datei anlegen** aus dem Menü **Datei** und bewegen den Cursor auf die neue Position in dem Programm oder Modul, indem Sie die Befehle zur Cursorbewegung verwenden.

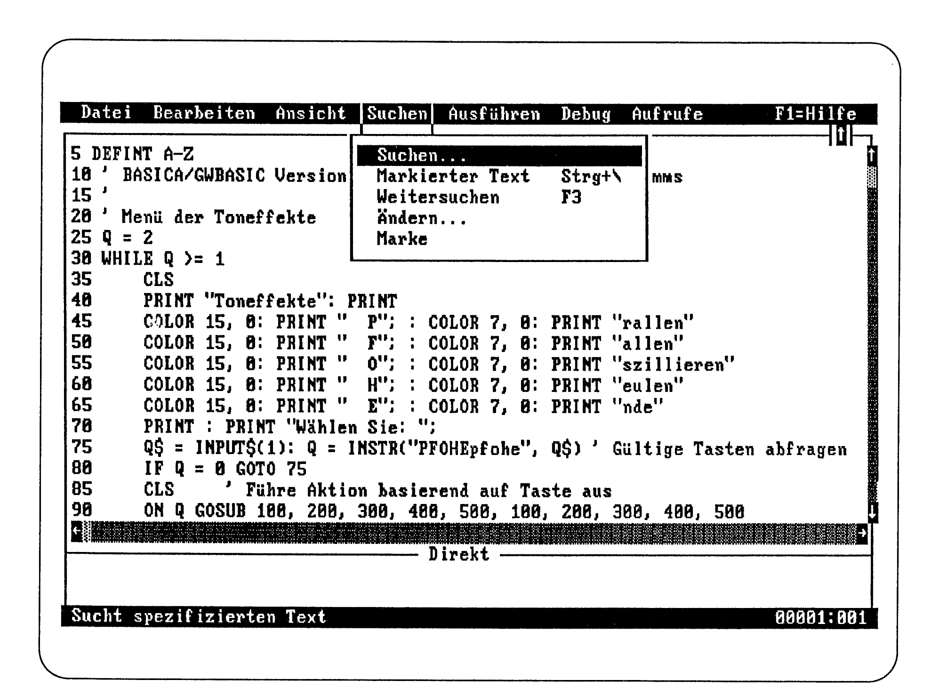
4. Wählen Sie **Einfügen** aus dem Menü **Bearbeiten** (oder betätigen Sie UMSCHALTTASTE+EINFG), um den Text einzufügen. Der Text verbleibt bis zum nächsten Befehl **Ausschneiden** oder **Kopieren** in der Zwischenablage, so daß Sie Text an mehr als einer Position einfügen können.

Das Kopieren eines Textblockes umfaßt im allgemeinen dieselbe Vorgehensweise: Sie müssen den Text, den Sie kopieren möchten, markieren, ihn in der Zwischenablage speichern, den Cursor auf die neue Position bewegen und den Text einfügen. Wenn Sie den markierten Text in der Zwischenablage speichern, sollten Sie anstelle des Befehls **Ausschneiden** (UMSCHALTTASTE+ENTF) den Befehl **Kopieren** (STRG+EINFG) verwenden.

5.8 Suchen und Ersetzen

Abbildung 5.2 zeigt das Menü **Suchen**, das es Ihnen ermöglicht, einen bestimmten Text überall in Ihrem Programm zu suchen und diesen wahlweise durch neuen Text zu ersetzen.

*Abbildung 5.2 Das Menü **Suchen***



Das Menü **Suchen** hat mehrere Verwendungen, besonders in langen und komplexen Programmen. Um sich zum Beispiel schnell zu einem entfernten Teil des Programms zu bewegen, können Sie nach einer Zeilenmarke oder anderen Zeichenkette suchen, von der Sie wissen, daß diese sich in jenem Abschnitt befindet, und anschließend mit der Bearbeitung des Programmtextes an dieser Stelle fortfahren. Außerdem können Sie nach einer Teilzeichenkette suchen und diese durch neuen Text ersetzen.

Um zu suchen oder zu ersetzen, müssen Sie QuickBASIC zunächst mitteilen, wonach es suchen soll. Zur Angabe des Ziels der Suche gibt es mehrere Möglichkeiten. Wenn sich der Cursor irgendwo auf einem Wort oder einem Leerzeichen, das dem Wort folgt, befindet, verwenden die Befehle aus dem Menü **Suchen** dieses Wort als Ziel der Suche. Wenn sich der Cursor aktuell auf einer leeren Zeile befindet, ist das letzte Wort, nach dem gesucht wurde, das Standardziel für die aktuelle Suche.

Eine Suche beginnt immer an der Cursor-Position und bewegt sich solange nach unten, bis der angegebene Text gefunden ist, oder der gesamte Text durchsucht ist, ohne eine Entsprechung zu finden. Nachdem eine Entsprechung einmal gefunden ist, haben Sie erneut mehrere Möglichkeiten: Sie können den Zieltext mit neuem Text ersetzen, die Suche fortsetzen, oder die Suche an dieser Stelle abbrechen.

Wenn eine Suche das Textende erreicht, ohne eine Entsprechung gefunden zu haben, wird die Suche, beginnend am Textanfang, fortgesetzt, bis eine Entsprechung gefunden wird, oder die aktuelle Cursor-Position erreicht ist. Somit können Sie mit der Bearbeitung des Programmtextes an der Cursor-Position fortfahren, wenn eine Suche in dem Text keine Entsprechung gefunden hat.

Die Abschnitte 5.8.1 bis 5.8.2 beschreiben ausführlich die Befehle des Menüs **Suchen**.

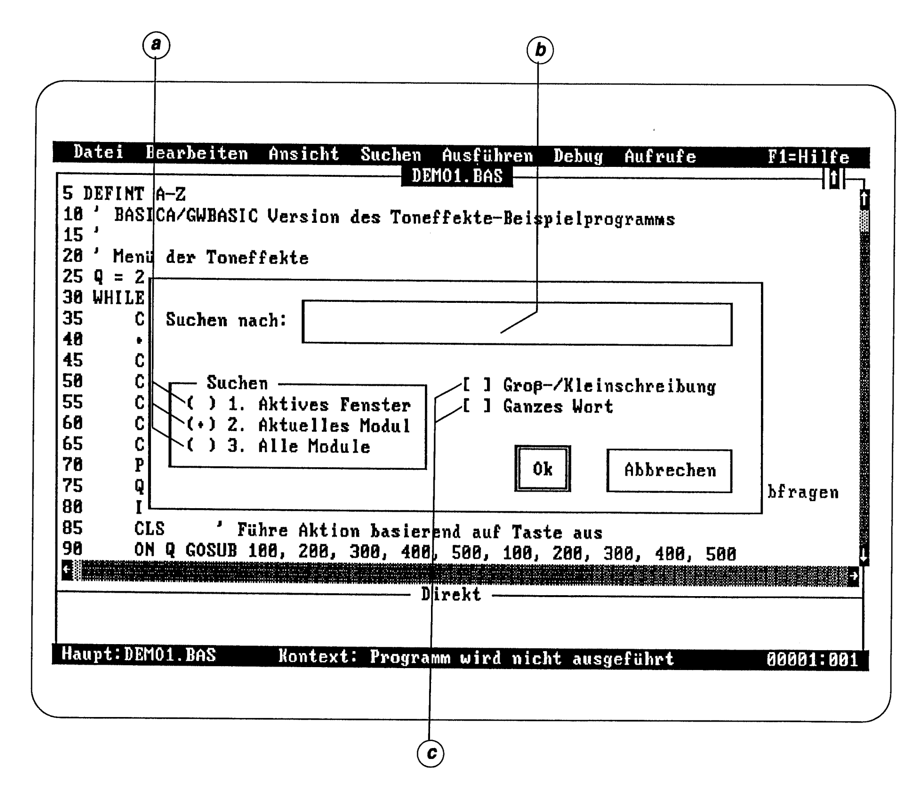
5.8.1 Wie Sie Text finden

QuickBASIC bietet verschiedene Möglichkeiten, das Ziel einer Suche anzugeben. Sie können nach einem einzelnen Zeichen, einem Wort oder einer Gruppe von Zeichen oder Wörter suchen. Darüber hinaus können Sie den Bereich der Suche definieren. Wenn das Programm mehr als ein Modul oder eine Prozedur hat, können Sie angeben, welches Modul oder welche Prozedur in die Suche miteinbezogen werden soll. Die folgenden Abschnitte beschreiben, wie die Befehle des Menüs **Suchen** verwendet werden, um markierten Text zu suchen oder zu verändern.

5.8.1.1 Suchen

Verwenden Sie den Befehl **Suchen** aus dem Menü **Suchen**, um in Ihrem Programm nach einem Zeichen, einem Wort oder einer Gruppe von Zeichen oder Wörter zu suchen. Wenn Sie den Befehl **Suchen** wählen, öffnet sich das Dialogfeld **Suchen** (siehe Abbildung 5.3).

Abbildung 5.3. Das Dialogfeld ***Suchen***



a) Geben Sie an, wo nach dem Text gesucht werden soll.

b) Geben Sie den zu suchenden Text ein.

c) Setzen Sie Einschränkungen bei der Suche.

Um Text zu finden, führen Sie die folgenden Schritte durch:

1. Geben Sie den zu suchenden Text in das Textfeld "Suchen nach" ein.
2. Verwenden Sie die Optionsflächen von **Suchen**, um anzugeben, wo nach dem Text gesucht werden soll. Die drei Optionen lauten wie folgt:
   - Die Option "Aktives Fenster" sucht nach Text nur in dem aktiven Fenster.
   - Die Option "Aktuelles Modul" sucht nach Text nur in dem aktuellen Modul.
   - Die Option "Alle Module" sucht nach Text in allen geladenen Modulen.
3. Setzen Sie Einschränkungen bei der Suche, falls notwendig, indem mit der Option "Ganzes Wort" oder der Option "Groß-/Kleinschreibung" oder beidem verglichen wird.

- Die Option "Ganzes Wort" findet den angegebenen Text nur, wenn dieser von Leerzeichen, Interpunktionszeichen oder anderen Zeichen, die nicht als Teil eines Wortes betrachtet werden, eingeschlossen ist. Die Zeichen A-Z, a-z, 0-9 und !, #, $, % und & (normalerweise in Variablendeklarationen verwendet) werden als Teile eines Wortes betrachtet. Wenn zum Beispiel CHR der angegebene Text ist, findet die Option "Ganzes Wort" CHR:, nicht jedoch CHR$.
- Die Suche mit der Option "Groß-/Kleinschreibung" ist nur dann erfolgreich, wenn der angegebene Text exakt mit dem gesuchten Text übereinstimmt. Wenn zum Beispiel CHR der angegebene Text ist, findet die Option "Groß-/Kleinschreibung" CHR, nicht jedoch Chr.

4. Betätigen Sie die EINGABETASTE, um den Befehl auszuführen.

Wenn der Text nicht gefunden wurde, erscheint die Meldung `Suchbegriff nicht gefunden`. Betätigen Sie die EINGABETASTE, um die Meldung zu entfernen und fortzufahren.

5.8.1.2 Markierter Text

Der Befehl **Markierter Text** aus dem Menü **Suchen** bietet Ihnen eine weitere Möglichkeit, nach Text zu suchen: Dieser Befehl sucht nach Text, der im aktuellen Fenster hervorgehoben ist. Folgen Sie diesen Schritten, um **Markierter Text** einzusetzen:

1. Markieren Sie den Text, nach dem Sie suchen möchten. Beachten Sie, daß der markierte Text komplett auf eine einzige Zeile passen muß.
2. Wählen Sie den Befehl **Markierter Text** aus dem Menü **Suchen** (oder betätigen STRG+\).

Die Hervorhebung bewegt sich zur nächsten Stelle, an der der markierte Text auftritt.

5.8.1.3 Weitersuchen

Der Befehl **Weitersuchen** aus dem Menü **Suchen** sucht nach dem Text, der in dem vorhergehenden Befehl **Suchen** angegeben wurde. Die Tastenkurzkombination ist F3.

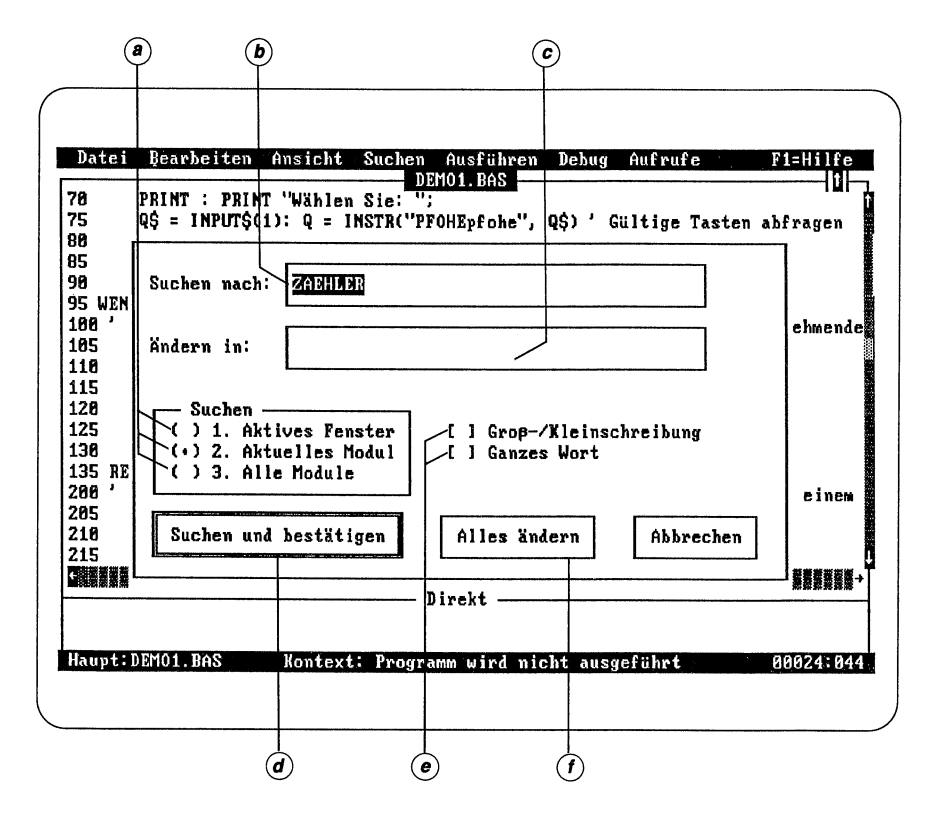
5.8.1.4 Marke

Der Befehl **Marke** aus dem Menü **Suchen** sucht nach einer Zeilenmarke, indem er einen Doppelpunkt an den aktuell markierten Text oder den Text, den Sie im vorhergehenden Befehl **Suchen** angegeben haben, anhängt. Wenn zum Beispiel das Wort Eiffelturm der gewählte Text ist, veranlaßt die Wahl **Marke** QuickBASIC dazu, nach der Zeilenmarke Eiffelturm: zu suchen.

5.8.2 Wie Sie markierten Text ersetzen (Ändern)

Verwenden Sie den Befehl **Ändern** aus dem Menü **Suchen**, um nach Text zu suchen und diesen mit neuem Text zu ersetzen. Wenn Sie den Befehl **Ändern** wählen, öffnet sich das in Abbildung 5.4 gezeigte Dialogfeld:

Abbildung 5.4 Das Dialogfeld ***Ändern***

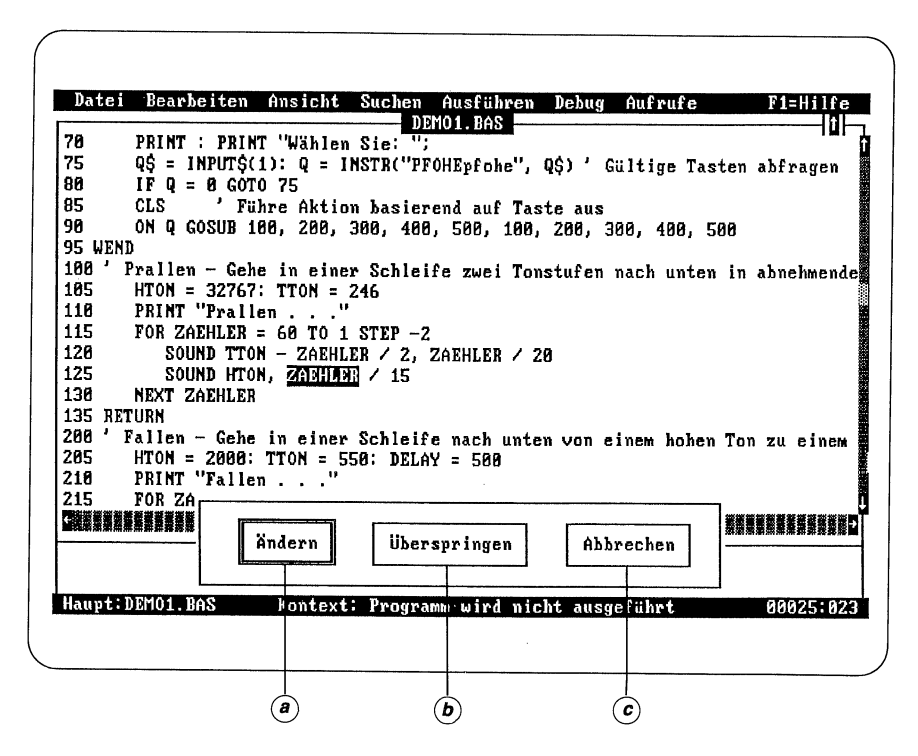


a) Geben Sie an, wo nach dem Text gesucht werden soll.
b) Geben Sie den zu suchenden Text ein.
c) Geben Sie den Text ein, der gefundenen Text ersetzt.
d) Bestätigung erbitten, bevor eine Änderung durchgeführt wird.
e) Setzen der Einschränkungen für die Suche.
f) Änderungen bei jedem Auftreten ohne Bestätigung durchführen.

Um Text zu ändern, führen Sie die folgenden Schritte aus:

1. Geben Sie den zu suchenden Text in das Textfeld "Suchen nach" ein.
2. Geben Sie den Ersatztext in das Textfeld "Ändern in" ein.
3. Verwenden Sie die Optionsflächen von **Suchen**, um anzugeben, wo nach dem zu ändernden Text gesucht werden soll. Die drei Optionen lauten wie folgt:
   - Die Option "Aktives Fenster" sucht nur im aktiven Fenster nach Text.
   - Die Option "Aktuelles Modul" sucht nur in dem aktuellen Modul nach Text.
   - Die Option "Alle Module" sucht in allen geladenen Modulen nach Text.
4. Setzen Sie Einschränkungen bei der Suche, falls notwendig, indem mit der Option "Ganzes Wort" oder der Option "Groß-/Kleinschreibung" oder beidem verglichen wird.
   - Die Option "Ganzes Wort" findet den angegebenen Text nur, wenn dieser von Leerzeichen, Interpunktionszeichen oder anderen Zeichen, die nicht als Teil eines Wortes betrachtet werden, eingeschlossen ist. Die Zeichen A-Z, a-z, 0-9 und !, #, $, % und & (normalerweise in Variablendeklarationen verwendet) werden als Teile eines Wortes betrachtet. Wenn zum Beispiel CHR der angegebene Text ist, findet die Option "Ganzes Wort" CHR:, nicht jedoch CHR$.
   - Die Suche mit der Option "Groß-/Kleinschreibung" ist nur dann erfolgreich, wenn der angegebene Text exakt mit dem gesuchten Text übereinstimmt. Wenn zum Beispiel CHR der angegebene Text ist, findet die Option "Groß-/Kleinschreibung" CHR, nicht jedoch Chr.
5. Führen Sie den Befehl aus oder brechen Sie diesen ab, indem Sie zwischen den Befehlsflächen "Suchen und bestätigen", "Alles ändern" oder "Abbrechen" wählen.
   - Die Befehlsfläche "Suchen und bestätigen" sucht den Text und zeigt dann drei weitere Befehlsflächen an: "Ändern", "Überspringen" und "Abbrechen". Die Befehlsfläche "Ändern" führt die Änderung aus, die Fläche "Überspringen" sucht nach dem nächsten Auftreten (ohne Änderungen durchzuführen), und die Fläche "Abbrechen" beendet die Suche und bringt Sie in das aktive Fenster zurück (siehe Abbildung 5.5).

Abbildung 5.5 Die Befehlsflächen Ändern, Überspringen und Abbrechen



a) Die Änderung wird durchgeführt.

b) Es wird nach dem nächsten Auftreten des Textes gesucht, ohne eine Änderung vorzunehmen.

c) Die Suche wird beendet, Sie kehren in das aktive Fenster zurück.

- Die Befehlsfläche "Alles ändern" sucht und ersetzt jedes Vorkommen des angegebenen Textes, ohne nach einer Bestätigung zu fragen.
- Die Befehlsfläche "Abbrechen" bricht den Befehl ab.

Wichtig Der Befehl **Rückgängig** aus dem Menü **Bearbeiten** kann eine Änderung, die Sie mit dem Befehl **Ändern** aus dem Menü **Suchen** durchführen, nicht rückgängig machen. Seien Sie bei Änderungen vorsichtig, insbesondere bei "Alles ändern", da diese Option drastische Änderungen in Ihrem Programm vornehmen kann.

5.9 Wie Sie Textmarkierungspunkte im Text verwenden

Wenn Sie an verschiedenen Teilen eines großen Programmes arbeiten, können Sie Textmarkierungspunkte an unterschiedlichen Stellen des Programmes setzen und anschließend zu jedem Textmarkierungspunkt springen, wenn Sie erneut in diesem Teil des Programms arbeiten müssen. Sie können bis zu vier Textmarkierungspunkte, numeriert von 0 bis 3, an beliebiger Programmstelle setzen. Um diese Textmarkierungspunkte einzusetzen

- Betätigen Sie STRG+K *n*, um den Textmarkierungspunkt mit der Nummer *n* zu setzen.
- Betätigen Sie STRG+Q *n*, um den Cursor zu dem Textmarkierungspunkt mit der Nummer *n* zu bewegen.

5.10 Wie Sie Sonderzeichen eingeben

QuickBASIC besitzt Prozeduren zur Eingabe der Literaldarstellungen von Sonderzeichen, die ASCII-Zeichen höherer Ordnung sowie die ASCII-Zeichenliterale, die mit Steuersequenzen verknüpft sind, umfassen. (Im allgemeinen macht das Einfügen solcher Zeichen Ihre Programme zwar lesbarer, aber weniger portabel. Zur Portabilität sollten Sie die Funktion **CHR$** verwenden.)

5.10.1 Wie Sie ASCII-Zeichen höherer Ordnung eingeben

ASCII-Zeichen höherer Ordnung sind diejenigen, die den Zahlen von 128 bis 255 entsprechen. Sie können diese direkt in QuickBASIC eingeben (das heißt, ohne Verwendung der Funktion **CHR$**), indem Sie die ALT-TASTE gedrückt halten, während Sie den entsprechenden Dezimal-Code auf dem Zehnerblock eingeben. Wenn Sie zum Beispiel die ALT-TASTE gedrückt halten, während Sie 205 eingeben, erhalten Sie an der Cursor-Position das Zeichen Doppelte Linie.

5.10.2 Wie Sie Steuerzeichen eingeben

Viele der ASCII-Zeichen niedriger Ordnung, die den Dezimalzahlen 1 bis 27 entsprechen, sind sowohl mit Zeichenliteralen als auch mit besonderen Funktionen verknüpft. Zum Beispiel ist das ASCII-Zeichen, das von ALT+25 dargestellt wird, sowohl mit dem Zeichen Pfeil NACH UNTEN (↓) als auch mit ^Y (Befehl zum Löschen der aktuellen Zeile) verknüpft. Angenommen, Sie möchten den Pfeil NACH UNTEN in eine Druckanweisung einfügen. Eine Möglichkeit, dies durchzuführen, besteht darin, dezimal 25 als Argument für die Funktion **CHR$** anzugeben. Sie können das Zeichenliteral jedoch auch selbst eingeben, indem Sie die folgende Methode einsetzen:

1. Halten Sie die STRG-TASTE gedrückt, während Sie den Buchstaben P betätigen.

   Es erscheint das Zeichen ^P auf der QuickBASIC-Statusleiste. Während dieses Zeichen angezeigt wird, können Sie eine Steuertastensequenz eingeben, um deren Zeichenliteral zu erzeugen.

2. Halten Sie die STRG-TASTE erneut gedrückt, betätigen Sie aber diesesmal den Buchstaben, der demjenigen Zeichenliteral entspricht, das Sie wünschen.

   Dies fügt das ASCII-Zeichenliteral (im Gegensatz zu der bestimmten Funktion der Steuersequenz) am Cursor ein.

Wenn Sie zum Beispiel das Zeichen Pfeil NACH UNTEN (↓) innerhalb der Anführungszeichen einer **PRINT**-Anweisung eingeben möchten, können Sie es als STRG+P+Y eingeben. Falls Sie versuchen, dieses Zeichen als STRG+Y oder als ALT+25 einzugeben, währenddessen ^P nicht auf der Statusleiste sichtbar ist, würde Ihre Zeile einfach gelöscht. Beachten Sie, daß Sie diese Technik nicht verwenden können, um die folgenden Zeichen einzufügen: ^J, ^L, ^M, ^U. Eine komplette Liste der ASCII-Zeichen finden Sie in Anhang A, "Tasturabfragecodes und ASCII-Zeichencodes", im *BASIC-Befehlsverzeichnis.*

5.11 Einrücken

Verwenden Sie die LEERTASTE oder die TAB-TASTE, um eine einzelne Textzeile einzurücken. Um einen Textblock einzurücken, markieren Sie die Zeilen, die Sie einrücken möchten, und betätigen Sie TAB.

Der QuickBASIC-Editor merkt sich das aktuelle Niveau der Einrückung und rückt jede neue Zeile automatisch ein. Das heißt, wenn Sie die EINGABETASTE betätigen, um eine neue Zeile zu beginnen, erscheint der Cursor sofort unter dem ersten Zeichen der Zeile darüber. Diese Eigenschaft ist hilfreich, wenn Sie mehrere eingerückte Zeilen eingeben, wie zum Beispiel den Körper einer **FOR**-Schleife. Betätigen Sie die POS1-TASTE, um die automatische Einrückung abzubrechen und den Cursor zurück zum linken Rand zu bewegen.

Falls Sie eine Zeile zu weit nach rechts eingerückt haben, können Sie die führenden Leerzeichen bis zur nächsten Einrückung entfernen, in dem Sie den Cursor in diese Zeile bewegen und die Tasten UMSCHALTTASTE+TAB drücken. Ein eventuell markierter Textblock wird dann vollständig nach links verschoben.

Die normale Tabulatorabstand beträgt acht Leerzeichen. Sie können den Tabulatorabstand wie folgt verändern:

1. Wählen Sie den Befehl **Optionen** aus dem Menü **Ansicht.**
2. Bewegen Sie die Eingabefixierung auf die Bildschirmoption "Tab-Abstand".
3. Geben Sie den neuen Tabulatorabstand ein.
4. Betätigen Sie die EINGABETASTE.

QuickBASIC verwendet einzelne Leerzeichen (und nicht das Tabulatorzeichen, ASCII 9), um das Niveau der Einrückung darzustellen. Die Option "Tab-Abstand" aus dem Dialogfeld **Optionen** des Menüs **Ansicht** setzt die Anzahl an Leerzeichen pro Einrückungsniveau.

Einige Texteditoren verwenden das Tabulatorzeichen, um bei der Speicherung von Textdateien mehrere Leerzeichen darzustellen. Der QuickBASIC-Editor behandelt Tabulatorzeichen in solchen Dateien auf eine der folgenden Arten:

1. Tabulatorzeichen innerhalb von Zeichenketten in Anführungszeichen erscheinen als das Zeichen, das für das ASCII-Zeichen 9 (o) in der ASCII-Tabelle im Anhang A, "Tastaturabfragecodes und ASCII-Zeichencodes", im *BASIC-Befehlsverzeichnis* gezeigt wird.
2. Tabulatorzeichen, die außerhalb von Anführungszeichen stehen, rücken den Text, der ihnen folgt, auf das nächste Einrückungsniveau ein.

Falls Sie versuchen, den Tabulatorabstand zu verändern, während ein solches Programm geladen ist, gibt QuickBASIC die folgende Fehlermeldung aus:

```
Tab-Abstände können während des Ladens nicht geändert werden
```

Auf diese Weise werden Sie daran gehindert, alte Quelldateien, die mit anderen Editoren erstellt wurden, unabsichtlich neu zu formatieren. Falls Sie eine Datei laden und die Einrückung nicht wünschen, können Sie die Einrückung zurücknehmen, indem Sie den folgenden Ablauf verwenden:

1. Wählen Sie den Befehl **Neues Programm** aus dem Menü **Datei**, um Ihre Datei zu löschen, ohne sie zu speichern.
2. Wählen Sie den Befehl **Optionen** aus dem Menü **Ansicht**, und setzen Sie die Option "Tab-Abstand" wie in der vorhergehenden Liste beschrieben.
3. Wählen Sie den Befehl **Programm laden** aus dem Menü **Datei**, und laden Sie Ihr Programm erneut. Wenn Ihr Programm erneut geladen ist, werden die Einrückungen die neue Einstellung der Option "Tab-Abstand" berücksichtigen.

Beachten Sie, daß der vorhergehende Ablauf nur mit alten Programmen funktioniert. Text, der innerhalb von QuickBASIC erstellt wurde, kann nicht auf diese Art umformatiert werden, da QuickBASIC niemals Tabulatorzeichen benutzt.

5.12 Wie Sie Zeilen zusammensetzen

Gehen Sie wie folgt vor, um zwei Textzeilen zusammenzusetzen:

1. Bewegen Sie den Cursor auf das erste Zeichen der zweiten Zeile.
2. Betätigen Sie die RÜCKTASTE.

5.13 Wie Sie Text aus anderen Dateien kopieren

Es gibt zwei Möglichkeiten, um den Inhalt anderer Dateien zu dem aktuellen Programm, dem Dokument oder der Include-Datei hinzuzufügen. Sie können entweder die gesamte Datei oder aber einen Teil der Datei in den aktuellen Text kopieren.

5.13.1 Wie Sie eine komplette Datei kopieren

Um eine komplette Datei in den aktuellen Text zu kopieren, verwenden Sie den Befehl **Zusammenführen** aus dem Menü **Datei**. Der neue Text wird oberhalb des Cursors eingefügt.

5.13.2 Wie Sie einen Teil einer Datei kopieren

Um einen Teil einer Datei in den aktuellen Text zu kopieren:

1. Wählen Sie den Befehl **Datei laden** aus dem Menü **Datei**, um die Datei zu laden, die den Text enthält, den Sie kopieren möchten.

   Nur das gerade geladene Modul wird angezeigt. Normalerweise ist es jedoch einfacher, Text aus einem Modul in ein anderes zu kopieren, wenn beide Module sichtbar sind. Um beide Module anzuzeigen

   a. Wählen Sie den Befehl **SUBs** aus dem Menü **Ansicht**.

   b. Markieren Sie den Namen des Moduls oder der Prozedur, die Sie anzeigen möchten.

   c. Wählen Sie die Befehlsfläche "Teil-Fenster" aus dem Dialogfeld **SUBs** (oder betätigen Sie ALT+S). Dieses führt dazu, daß der Arbeitsbereich in zwei Teile geteilt wird, mit der Datei, die Sie zuvor geladen haben, in dem einen Fenster und dem Modul oder der Prozedur, die Sie gerade markiert haben, in dem anderen Fenster.

2. Betätigen Sie F6 oder UMSCHALTTASTE+F6, um den Cursor in das Fenster zu bewegen, das den Text enthält, den Sie kopieren möchten.
3. Markieren Sie den Text, den Sie kopieren möchten.
4. Wählen Sie die Befehle **Ausschneiden** oder **Kopieren** aus dem Menü **Bearbeiten** (oder betätigen Sie UMSCHALTTASTE+ENTF oder STRG+EINFG), um den markierten Text in der Zwischenablage zu speichern.
5. Betätigen Sie F6 oder UMSCHALTTASTE+F6, um den Cursor zurück in das Modul zu bewegen, in das Sie den neuen Text kopieren möchten.
6. Wählen Sie den Befehl **Einfügen** aus dem Menü **Bearbeiten** (oder UMSCHALTTASTE+EINFG). Der Text wird oberhalb des Cursors eingefügt.
7. Wählen Sie den Befehl **Datei entfernen** aus dem Menü **Datei**, um die Datei zu entfernen, aus der Sie den Text kopiert haben.

5.14 Zusammenfassung der Editierbefehle

Der QuickBASIC-Editor ist so gestaltet, daß Sie flexibel arbeiten können; er bietet eine Tastaturschnittstelle, die Benutzern anderer Microsoft-Produkte, wie zum Beispiel Microsoft Word, vertraut ist. Viele Kombinationen der STRG-TASTE mit anderen Tasten sind Benutzern von WordStar und ähnlichen Editoren geläufig. Die folgende Tabelle faßt alle Editierbefehle von QuickBASIC zusammen. Die Abschnitte 5.3 bis 5.11 beschreiben, wie die hier zusammengefaßten Funktionen kombiniert werden müssen, um komplexe Editieraufgaben durchführen zu können, wie zum Beispiel das Bewegen von Textblöcken.

Cursor bewegen	***Tasten der Tastatur***	***Tastenkombinationen im WordStar-Stil***
Zeichen links	NACH LINKS (←)	STRG+S
Zeichen rechts	NACH RECHTS (→)	STRG+D
Wort links	STRG+NACH LINKS (←)	STRG+A
Wort rechts	STRG+NACH RECHTS (→)	STRG+F
Zeile nach oben	NACH OBEN (↑)	STRG+E
Zeile nach unten	NACH UNTEN (↓)	STRG+X
Zeilenanfang	POS1	STRG+Q S
Anfang der nächsten Zeile	STRG+EINGABETASTE	STRG+J
Zeilenende	ENDE	STRG+Q D

Cursor bewegen	*Tasten der Tastatur*	*Tastenkombinationen im WordStar-Stil*
Fenster oben	---	STRG+Q E
Fenster unten	---	STRG+Q X
Anfang eines Moduls/einer Prozedur	STRG+POS1	STRG+Q R
Ende eines Moduls/einer Prozedur	STRG+ENDE	STRG+Q C
Markierungspunkte setzen	---	STRG+K *n**
Zum Markierungspunkte bewegen	---	STRG+Q *n**

Einfügen	*Tasten der Tastatur*	*Tastenkombinationen im WordStar-Stil*
Einfügemodus ein- oder ausschalten	EINFG	STRG+V
Zeile darunter	ENDE EINGABETASTE	---
Zeile darüber	---	POS1 STRG+N
Inhalt der Zwischenablage	UMSCHALTTASTE+EINFG	---
Tabulator an der Cursor-Position oder am Anfang jeder markierten Zeile	TAB	STRG+I
Steuerzeichen an der Cursor-Position	---	STRG+P STRG+*Taste***

Löschen	*Tasten der Tastatur*	*Tastenkombination im WordStar-Stil*
Aktuelle Zeile, in der Zwischenablage speichern	---	STRG+Y
Bis zum Ende der Zeile, in der Zwischenablage speichern	---	STRG+Q Y
Zeichen links	RÜCKTASTE	STRG+H
Zeichen über dem Cursor	ENTF	STRG+G
Wort	---	STRG+T
Markierter Text, nicht in der Zwischenablage speichern	ENTF	STRG+G
Markierter Text, in der Zwischenablage speichern	UMSCHALTTASTE+ENTF	---
Führende Leerzeichen einer Einrückungsebene markierter Zeilen	UMSCHALTTASTE+TAB	---

Kopieren	***Tasten der Tastatur***	***Tastenkombination im WordStar-Stil***
Markierter Text, in der Zwischenablage speichern	STRG+EINFG	---

Rollen	***Tasten der Tastatur***	***Tastenkombination im WordStar-Stil***
Eine Zeile nach oben	NACH OBEN (↑)	STRG+W
Eine Zeile nach unten	NACH UNTEN (↓)	STRG+Z
Seite nach oben	BILD ↑	STRG+R
Seite nach unten	BILD ↓	STRG+C
Ein Fenster nach links	STRG+BILD ↑	---
Ein Fenster nach rechts	STRG+BILD ↓	---

Markieren	***Tasten der Tastatur***	***Tastenkombination im WordStar-Stil***
Zeichen links	UMSCHALTTASTE+ NACH LINKS (←)	---
Zeichen rechts	UMSCHALTTASTE+ NACH RECHTS (→)	---
Aktuelle Zeile	UMSCHALTTASTE+ NACH UNTEN (↓)	---
Zeile darüber	UMSCHALTTASTE+ NACH OBEN (↑)	---
Wort links	UMSCHALTTASTE+STRG+ NACH LINKS (←)	---
Wort rechts	UMSCHALTTASTE+STRG+ NACH RECHTS (→)	---
Bildschirm nach oben	UMSCHALTTASTE+BILD ↑	---
Bildschirm nach unten	UMSCHALTTASTE+BILD ↓	---
Bis zum Anfang eines Moduls/einer Prozedur	UMSCHALTTASTE+ STRG+POS1	---
Bis zum Ende eines Moduls/einer Prozedur	UMSCHALTTASTE+ STRG+ENDE	---

* Ersetzen Sie *n* durch eine Zahl von 0 bis 3, um den gewünschten Markierungspunkt zu kennzeichnen.

** Der Kombination STRG+P folgen STRG sowie die gewünschte Taste.

6 Wie Sie QuickBASIC-Programme erstellen und ausführen

Ob Sie nun bisher mit BASIC-Interpretern oder mit BASIC-Compilern gearbeitet haben, die vorhergehenden Kapitel bieten Ihnen alle notwendigen Informationen, um Programme in QuickBASIC zu schreiben. Dieses Kapitel erläutert den Ablauf der Programmierung innerhalb der QuickBASIC-Umgebung und zeigt effiziente Möglichkeiten auf, die Leistungsfähigkeit von QuickBASIC einzusetzen.

Wenn Sie dieses Kapitel beendet haben, werden Sie wissen, wie Sie

- In QuickBASIC Programmanweisungen eingeben.
- Das Direkt-Fenster einsetzen, um Ideen zu testen und Daten eines laufenden Programmes zu verändern.
- Argumente der Befehlszeile innerhalb von QuickBASIC verändern.
- **SUB**- und **FUNCTION**-Prozeduren erstellen.
- Die **DECLARE**-Anweisungen, die QuickBASIC beim Speichern Ihres Programmes erzeugt, interpretieren.
- Module zusätzlich zum Hauptmodul verwenden.
- Module in einem kompletten Programm zusammenfassen und anschließend dieses Programm ausführen.

6.1 Wie Sie ein Programm schreiben

QuickBASIC unterscheidet sich von anderen BASIC-Produkten in vielerlei Hinsicht. Viele dieser Unterschiede sind Erweiterungen der Sprache BASIC selbst. Der grundlegende Unterschied zwischen QuickBASIC und anderen Programmiersprachen liegt jedoch darin, *wie* Sie es zum Schreiben Ihrer Programme einsetzen. QuickBASIC besitzt einen größeren Umfang an Programmiereigenschaften als frühere Programm-Entwicklungsumgebungen, und alle seine Eigenschaften sind komplett integriert. Das heißt, Sie müssen eine Tätigkeit nicht beenden, um etwas anderes durchzuführen. Zum Beispiel können Sie ein Programm debuggen, während Sie es bearbeiten.

Die folgende Liste enthält weitere Beispiele dieser nahtlosen Integration:

- Die Ausführung dessen, was Sie geschrieben haben, zu jedem beliebigen Zeitpunkt, unabhängig davon, wie vollständig oder unvollständig dieses ist, einfach durch Betätigen einer Taste. Da jede Anweisung in ausführbaren Code übersetzt wird, während Sie diese eingeben, gibt es keinen separaten Kompilierschritt innerhalb der QuickBASIC-Umgebung.
- Unterbrechen eines laufenden Programmes, Verwenden von Debug-Befehlen und -Techniken, um das Verhalten des Programmes zu analysieren, Bearbeiten des Programmes und anschließendes Fortfahren an dem Punkt, an dem Sie es angehalten haben.
- Schreiben separater Programmodule, diese einzeln austesten und anschließend zusammen laden, und sie als ein einziges Programm ausführen.

Hinweis Sobald Ihr Programm einmal fehlerfrei ist und zu Ihrer Zufriedenheit läuft, können Sie den Befehl **EXE-Datei erstellen** aus dem Menü **Ausführen** verwenden, um eine selbständig ausführbare Version Ihres Programms zu erstellen. In diesem Fall werden Sie feststellen, daß es einen separaten Kompilierschritt gibt, bei dem Ihr Programm auch gebunden wird. Eine Erläuterung, wie ausführbare Dateien innerhalb von QuickBASIC erstellt werden, finden Sie in Abschnitt 6.2. Weitere Informationen darüber, wie Sie außerhalb der QuickBASIC-Umgebung kompilieren, finden Sie in Kapitel 9, "Wie Sie aus DOS heraus kompilieren und binden".

6.1.1 Wie Sie ein Hauptmodul anlegen

In einem einmoduligen Programm ist das Hauptmodul das Programm. Sie haben also, sobald Sie Ihre erste Anweisung eingetippt und gespeichert haben, Ihr erstes Hauptmodul erstellt. In einem mehrmoduligen Programm enthält das Hauptmodul die erste Anweisung, die nach dem Start des Programms ausgeführt wird.

Wenn Sie QuickBASIC ohne Angabe eines Dateinamens starten, enthält die Titelleiste des Arbeitsbereiches das Wort "Unbenannt". Wenn Sie den Inhalt des Arbeitsbereiches speichern, wird der Name, unter dem Sie den Arbeitsbereich speichern, selbst dann, wenn dieser leer ist, der Name des Programm-Hauptmoduls.

Nachdem Sie das Hauptmodul benannt haben, erscheint der Name in der Titelleiste des Arbeitsbereiches. Darüber hinaus erscheint der Name auf der linken Seite der Statuszeile unten auf Ihrem Bildschirm, direkt hinter dem Wort Haupt.

6.1.2 Wie Sie Programmanweisungen eingeben

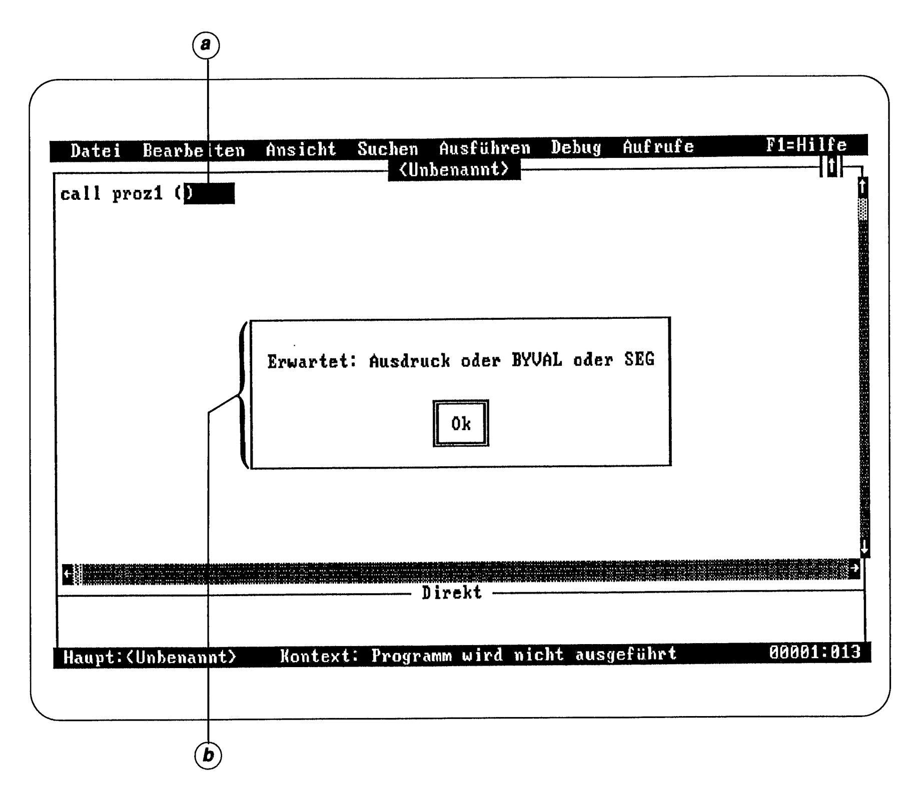
Mit QuickBASIC können Sie Ihre Programmanweisungen genauso eingeben, wie Sie es mit einem Texteditor oder einem BASIC-Interpreter tun würden. Jedesmal jedoch, wenn Sie eine eingetippte Zeile verlassen, überprüft BASIC diese auf Syntaxfehler. Wenn keine Syntaxfehler vorliegen, übersetzt QuickBASIC die Anweisung in ausführbaren Code. Wenn Syntaxfehler vorhanden sind, gibt QuickBASIC ein Dialogfeld aus, das eine entsprechende Fehlermeldung enthält.

Häufig werden Sie die Fehlermeldung `Syntaxfehler` sehen, einige Fehlermeldungen aber sind spezieller. Wenn Sie zum Beispiel aus Versehen die folgende Zeile eingeben:

```
CALL proc1 ()
```

erzeugt QuickBASIC die Fehlermeldung `Erwartet: Ausdruck oder BYVAL oder SEG` (siehe Abbildung 6.1).

Abbildung 6.1 Fehlermeldung

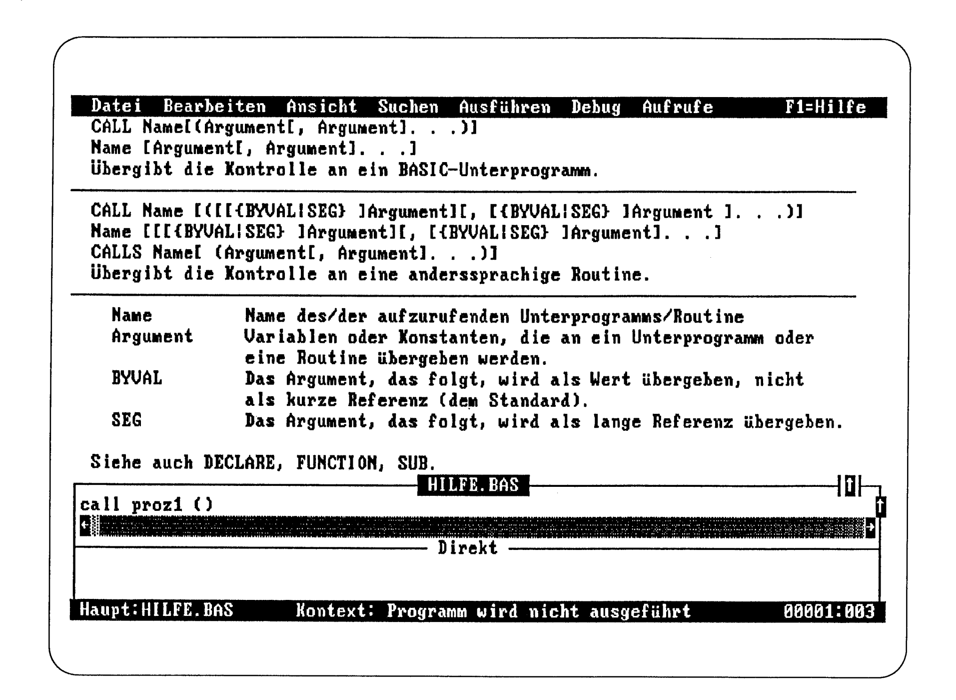


a) Die Hervorhebung zeigt, wo der Fehler auftrat.
b) Das Dialogfeld enthält die Fehlerbeschreibung.

Die Ursache für die Fehlermeldung liegt darin, daß auf eine **SUB**-Prozedur folgende leere Klammern keine gültige Syntax in einer **CALL**-Anweisung sind (obwohl diese in einer **DECLARE**-Anweisung absolut gültig sind). Wenn dem Namen Klammern folgen, erwartet QuickBASIC, daß diese ein Argument enthalten, entweder in der Form eines Ausdruckes oder als Argument, dem das Schlüsselwort **BYVAL** oder **SEG** vorausgeht. Die Fehlermeldungen in Dialogfeldern hängen davon ab, was Sie gerade eintippen, so daß, wenn Sie einen Tippfehler machen oder die Syntax zweier verwandter Anweisungen durcheinanderbringen, die daraus folgende Fehlermeldung zunächst verwirrend erscheint.

Wenn Sie den Cursor im obigen Fall irgendwo auf dem Schlüsselwort **CALL** plazieren und anschließend UMSCHALTTASTE+F1 betätigen, gibt QuickBASIC einen entsprechenden Hilfetext – auch kontext-sensitive Hilfe genannt – auf dem Bildschirm aus, der die Syntax der **CALL**-Anweisung zeigt (siehe Abbildung 6.2).

Abbildung 6.2 Kontext-sensitive Hilfe für die Anweisung ***CALL***



Die Syntax zeigt die Schlüsselwörter **BYVAL** und **SEG**, die in dem Zusammenhang des Aufrufs einer in einer anderen Sprache geschriebenen Prozedur verwendet werden. Wenn es dies nicht war, was Sie versucht haben, besteht die einzige andere Möglichkeit in einem Problem mit den Klammern. Die allgemeine Syntax im oberen Abschnitt des Bildschirms zeigt Klammern und Argumente als optional, wenn Sie aber Klammern verwenden, werden die Argumente als zwingend dargestellt. Das Entfernen der Klammern beseitigt also den Fehler.

Wenn QuickBASIC eine Fehlermeldung anzeigt, die verwirrend erscheint, sollten Sie generell die kontext-sensitive Hilfe verwenden; oder ziehen Sie das *BASIC-Befehlsverzeichnis* zu Rate, und werten Sie dann die Fehlermeldung aus, indem Sie das, was Sie eingetippt haben, mit der Syntax der Anweisung vergleichen.

6.1.3 Wie Sie im Speicher übersetzen und ausführen

Um Code zu testen, den Sie in den Arbeitsbereich eingegeben haben, wählen Sie den Befehl **Start** aus dem Menü **Ausführen** (oder betätigen Sie UMSCHALTTASTE+F5). Da von QuickBASIC jede Zeile während der Eingabe in ausführbaren Code übersetzt wird, übersetzt es den verbleibenden Rest einzelner Zeilen sehr schnell. Da alles im Speicher durchgeführt wird, muß QuickBASIC darüber hinaus nicht auf Diskettendateien zugreifen.

Auch wenn jede einzelne Zeile Ihres Programms ohne jeden Fehler übersetzt wird, können Sie möglicherweise immer noch Fehlermeldungen erhalten, wenn Sie **Start** aus dem Menü **Ausführen** wählen, um Ihr Programm zu starten. Zum Beispiel könnte einer **FOR**-Anweisung eine passende **NEXT**-Anweisung zum Abschluß der Schleife fehlen, oder Sie könnten versuchen, ein unzulässiges Argument an eine BASIC-Funktion zu übergeben. QuickBASIC entdeckt diese Art von Fehlern nicht, während Sie Ihr Programm eingeben, weil diese gültige Syntax darstellen. Diese Fehler treten nur auf, wenn Sie versuchen, Ihr Programm zu starten. Schlagen Sie in Anhang D, "Fehlermeldungen", im *BASIC-Befehlsverzeichnis* nach, wenn Sie bei dem Versuch, Ihr Programm zu starten, eine andere Fehlermeldung als "Syntaxfehler" erhalten.

6.1.4 Wie Sie zum Ausgabebildschirm zurückkehren (Ausgabebildschirm)

Wenn Sie Code ausführen, der Bildschirmausgaben beinhaltet, gibt QuickBASIC die Meldung `Weiter mit jeder beliebigen Taste...` in der letzten Zeile des Ausgabebildschirms aus. Nachdem Sie eine Taste betätigt haben und in den QuickBASIC-Editor zurückgekehrt sind, können Sie zwischen dem Editor und dem Ausgabebildschirm hin- und herschalten, indem Sie den Befehl **Ausgabebildschirm** aus dem Menü **Ansicht** wählen (oder F4 betätigen).

6.1.5 Wie Sie Ideen mit dem Direkt-Fenster testen

Das Direkt-Fenster ist vergleichbar mit dem "Direktmodus" in BASICA. Im Direktmodus von BASICA können Sie eine einzelne Zeile eingeben, die eine oder mehrere Anweisung/en enthält, und anschließend die EINGABETASTE betätigen, um diese Zeile auszuführen. Dies gibt Ihnen die Möglichkeit, Anweisungen direkt auszuführen, ohne diese Teil eines Programmes werden zu lassen.

Während Sie Ihr Programm schreiben, können Sie sich durch Betätigung von F6 in das Direkt-Fenster bewegen. Sobald Sie sich einmal in dem Direkt-Fenster befinden, können Sie Anweisungen eintippen und bearbeiten, wie Sie es auch im Arbeitsbereich tun, und diese anschließend sofort ausführen. Das Direkt-Fenster bietet eine gute Möglichkeit, Ideen für die Programmierung zu testen, bevor Sie diese Teil Ihres Programmes werden lassen.

6.1.5.1 Merkmale des Direkt-Fensters

Das Direkt-Fenster besitzt die folgenden Merkmale:

- Sie können jede Anweisung oder Gruppe von Anweisungen in einer Zeile ausführen, indem Sie den Cursor an beliebiger Stelle auf der Zeile plazieren und die EINGABETASTE betätigen.
- Sie können bis zu 10 Zeilen in das Direkt-Fenster eingeben, anschließend zwischen diesen hin- und herspringen und sie in beliebiger Reihenfolge ausführen.

  Nachdem Sie 10 Zeilen eingegeben haben, rollt jede neue Zeile den Text in dem Direkt-Fenster um eine Zeile nach oben.
- Eine Zeile kann maximal 256 Zeichen enthalten. Mehrere Anweisungen können in einer Zeile vorkommen, müssen aber durch Doppelpunkte (:) voneinander getrennt werden.
- Jede Zeile wird unabhängig von allen anderen Zeilen in dem Fenster ausgeführt. Dennoch kann der Inhalt einer Zeile andere Zeilen beeinflussen. Zum Beispiel enthalten die folgenden Zeilen zwei Zuweisungen für die Ganzzahlvariable `x%`, so daß der von der zweiten Zeile ausgegebene Wert derjenigen Anweisung entspricht, die zuletzt ausgeführt wurde:

  ```
  x% = 5
  print x%
  x% = 9
  ```

- Sie können das Direkt-Fenster vergrößern, indem Sie wiederholt ALT+PLUS betätigen, während es das aktive Fenster ist. Die Betätigung von STRG+F10, während das Direkt-Fenster aktiv ist, dehnt dieses auf den gesamten Bildschirm aus, aber ganz gleich, wie viele Zeilen sich dort befinden, Sie können nur die ersten 10 Zeilen verwenden.
- Code, der in das Direkt-Fenster geschrieben ist, kann nicht auf Diskette gespeichert werden, es sei denn, Sie verwenden die Befehle **Ausschneiden** oder **Kopieren** aus dem Menü **Bearbeiten**, um den Code in den Arbeitsbereich zu bewegen, so daß Sie ihn zusammen mit der restlichen Datei abspeichern können.

6.1.5.2 Anweisungen, die in dem Direkt-Fenster zulässig sind

Mit Ausnahme der in der weiter unten aufgeführten Liste, sind die meisten Anweisungen in dem Direkt-Fenster zulässig. Zum Beispiel können Sie eine gesamte **FOR...NEXT**-Schleife auf eine Zeile schreiben:

```
FOR i% = 1 TO 100 : ? i%, SQR(i%) : NEXT i%
```

Die folgende Liste zeigt die QuickBASIC-Schlüsselwörter, die Sie nicht im Direkt-Fenster verwenden können:

COMMON	**END SUB**
CONST	**END TYPE**
DATA	**FUNCTION**
DECLARE	**$INCLUDE**
DEF FN	**OPTION**
DEF*Typ*	**REDIM**
DIM	**SHARED**
$DYNAMIC	**$STATIC**
ELSEIF	**STATIC**
END DEF	**SUB**
END FUNCTION	**TYPE**
END IF	

Hinweis Obwohl die Anweisung **END IF** in dem Direkt-Fenster nicht zulässig ist, können Sie dort noch immer die einzeilige Form von **IF...THEN...ELSE** verwenden.

6.1.5.3 Wie Sie Berechnungen durchführen

Sie können das Direkt-Fenster auch dazu einsetzen, komplizierte Ausdrücke zu berechnen und das Ergebnis anschließend mit einer **PRINT**-Anweisung (oder dem Kurzzeichen ?, das für **PRINT** steht - genauso wie in BASICA) auszugeben. Sie können jede der in BASIC eingebauten Funktionen, wie zum Beispiel **SIN** oder **EXP**, in diesen Berechnungen verwenden.

Darüber hinaus können Sie das Direkt-Fenster dazu verwenden, den von **FRE(-1)** zurückgegebenen Wert ausdrucken zu lassen. Dies gibt Ihnen den Betrag des verfügbaren Speichers an, wenn sowohl QuickBASIC als auch Ihr Programm geladen ist.

6.1.5.4 Wie Sie den Ausgabebildschirm prüfen

Während ein Programm an Umfang und Komplexität zunimmt, können Sie Code in das Direkt-Fenster schreiben und diesen dort zuerst testen, bevor Sie ihn in Ihr Programm einbinden. Es wird damit vermieden, daß Sie ein Programm jedesmal, wenn Sie nur einen kleinen Teil des Codes verändern möchten, vom Beginn an neu starten müssen.

Beispiel

Sie können die folgenden Zeilen in dem Direkt-Fenster dazu verwenden, die Position von Ausgabe, die auf den Bildschirm geschrieben werden soll, zu testen:

```
cls : zei% = 1 : spa% =4*zei% : LOCATE zei%,spa% : PRINT "."
```

Indem Sie den Wert der Variablen `zei%` verändern und anschließend die EINGABETASTE betätigen, können Sie die Ausgabe der **PRINT**-Anweisung an unterschiedlichen Stellen des Bildschirms positionieren.

Manchmal kann es vorkommen, daß ein Programm den Ausgabebildschirm in einem unerwünschten Modus beläßt, wenn es die Ausführung beendet (zum Beispiel könnte ein Grafikprogramm enden, ohne den Bildschirm auf den Textmodus mit 80 Spalten zurückzusetzen). Sollte dies der Fall sein, verwenden Sie das Direkt-Fenster dazu, den Bildschirm in den von Ihnen gewünschten Modus zu bringen. Die folgenden Zeilen setzen zum Beispiel den Ausgabebildschirm wieder in den Textmodus mit 80 Spalten und schließen jedes Textfenster (ein begrenzter horizontaler Ausschnitt des Bildschirms), das in dem Programm gesetzt wurde:

```
screen 0
width 80
view print
```

6.1.5.5 Wie Sie Prozeduren aufrufen

Sie können die Auswirkungen einer beliebigen einzelnen **SUB**- oder **FUNCTION**-Prozedur isolieren, indem Sie diese aus dem Direkt-Fenster aufrufen. Verwenden Sie das Direkt-Fenster, um nur solche Anweisungen auszuführen, die zwischen den Anweisungen **SUB...END SUB** oder **FUNCTION...END FUNCTION** stehen. Wenn Sie beispielsweise folgende Zeile im Direkt-Fenster eingeben, werden alle Anweisungen in der Prozedur `UnterPro1`, inklusive aller Aufrufe anderer Prozeduren ausgeführt:

```
call UnterPro1
```

6.1.5.6 Wie Sie Variablen im laufenden Programm verändern können

Sie können ein Programm bis zu einem bestimmten Punkt ausführen, dann unterbrechen und mit dem Direkt-Fenster einer Variable einen neuen Wert zuweisen. Der Wert, den Sie in dem Direkt-Fenster zuweisen, wird der Wert der Variablen in dem laufenden Programm.

Beispiel

Die folgenden Schritte zeigen, wie Sie das Direkt-Fenster dazu einsetzen können, in einem laufenden Programm eine Variable zu verändern:

1. Geben Sie dieses Programm in den Arbeitsbereich ein:

```
FOR i% = 0 TO 10000
    LOCATE 10, 20 : PRINT i%
    LOCATE 10, 27 : PRINT "   ich zähle noch"
NEXT i%
```

2. Starten Sie das Programm, und verwenden Sie anschließend STRG+UNTBR, um es zu unterbrechen.
3. Geben Sie die folgende Zeile in das Direkt-Fenster ein:

```
i% = 9900
```

4. Betätigen Sie F5, um das Programm fortzusetzen.

   Die Schleife wird nun von 9.900 bis 10.000 ausgeführt.

Der Kontext jeder Zeile mit Anweisungen, die in dem Direkt-Fenster ausgeführt werden, ist derjenige der im aktiven Arbeitsbereich hervorgehobenen Anweisung. Wenn zum Beispiel die Anweisung, die Sie gerade ausgeführt haben, einer `x%` benannten Variablen einen Wert von 3 gab, und Sie

```
PRINT x%
```

in das Direkt-Fenster eingeben, wird die Zahl 3 auf den Bildschirm ausgegeben, wenn Sie die EINGABETASTE betätigen.

Ähnlich verhält es sich, wenn die nächste in dem Arbeitsbereich auszuführende Anweisung eine Anweisung auf Modul-Ebene ist. Sie können aus dem Direkt-Fenster heraus nicht auf die Variablen einer **SUB** oder **FUNCTION** zugreifen.

6.1.5.7 Wie Sie Laufzeitfehler simulieren

Laufzeitfehler, die während eines Programmlaufes auftreten, sind mit einem numerischen Code verknüpft. Eine Möglichkeit herauszufinden, welche Fehlermeldung mit einem gegebenen Code verknüpft ist, besteht darin, in Anhang D, "Fehlermeldungen", im *BASIC-Befehlsverzeichnis* nachzuschlagen. Eine weitere Möglichkeit besteht darin, den Fehler selbst zu simulieren. Sie können einen Fehler simulieren, indem Sie sich in das Direkt-Fenster bewegen und die gegebene Nummer als Argument für die Anweisung **ERROR** eingeben. QuickBASIC zeigt dann das Dialogfeld für diesen Fehler an, als wäre dieser Fehler aktuell in einem laufenden Programm aufgetreten.

6.1.6 Programme mit Argumenten auf der Befehlszeile (Ändere COMMAND$)

Viele Programme sind so geschrieben, daß sie Zeichenketten verarbeiten, die ihnen von der DOS-Befehlszeile übergeben werden. Die Funktion **COMMAND$** ermöglicht es einem Programm, auf diese Zeichenketten der Befehlszeile zuzugreifen.

Mit dieser QuickBASIC-Version können Sie jede an ein Programm übergebene Zeichenkette verändern, indem Sie den Befehl **Ändere COMMAND$** aus dem Menü **Ausführen** wählen. Alles das, was Sie in das Dialogfeld eingeben, wird zu der Zeichenkette, die von der Funktion **COMMAND$** zurückgegeben wird.

Weitere Informationen zu Argumenten auf der Befehlszeile sowie der Funktion **COMMAND$** finden Sie im *BASIC-Befehlsverzeichnis*.

6.2 Wie Sie innerhalb von QuickBASIC ausführbare Dateien erstellen (EXE-Datei erstellen)

Nachdem Ihr Programm einmal innerhalb von QuickBASIC läuft, können Sie eine Programmversion erstellen, die direkt von der DOS-Befehlszeile aus ohne Unterstützung von QuickBASIC läuft. Die Datei eines solchen Programmes wird als ausführbare Datei bezeichnet und hat die Erweiterung .EXE. Durch Eingabe des Basisnamens einer ausführbaren Datei hinter der DOS-Anfrage wird das Programm gestartet.

Wenn Sie eine ausführbare Datei anlegen, verwendet QuickBASIC zunächst den BASIC-Befehlszeilen-Compiler BC, um Ihren BASIC-Quellcode in eine Zwischendatei zu übersetzen, die als Objektdatei bezeichnet wird. Anschließend verwendet QuickBASIC ein Programm, das Microsoft Overlay Linker (LINK) heißt, um alle separat kompilierten Module Ihres Programms in einer einzigen ausführbaren Datei zu vereinigen. LINK faßt ebenso die kompilierten Objektdateien, die von BC erstellt wurden, mit den Unterstützungsroutinen zusammen, die Ihr Programm für eine richtige Ausführung benötigt. Diese Routinen sind in den Laufzeitbibliotheken BRUN40.LIB oder BCOM40.LIB zu finden.

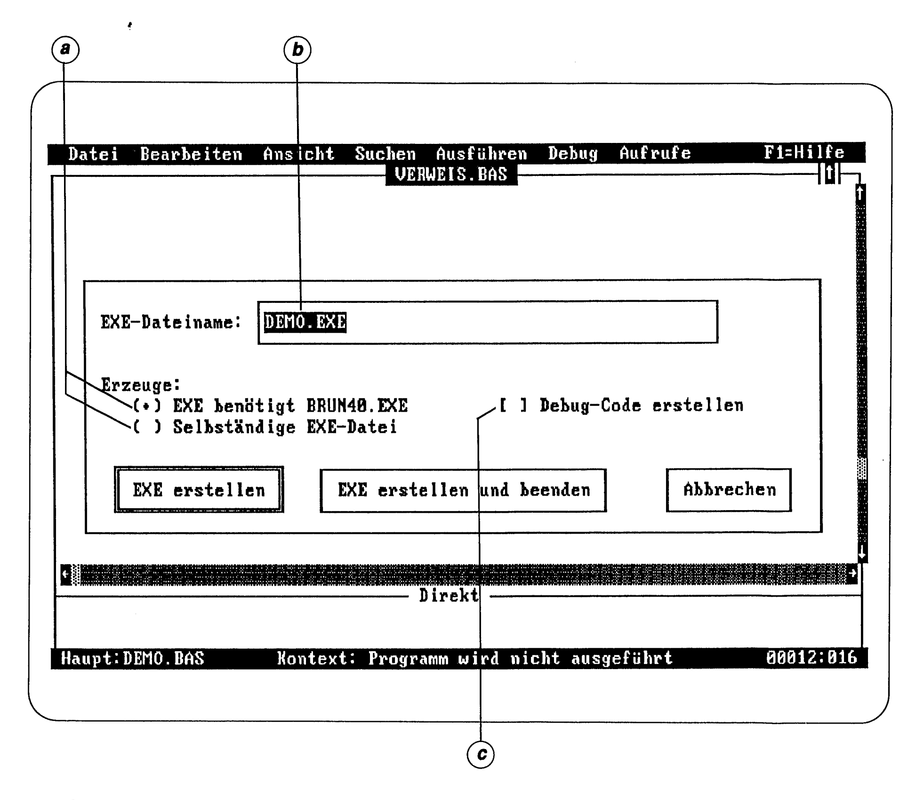
Die folgenden Dateien müssen für QuickBASIC verfügbar sein, wenn Sie eine ausführbare Datei erstellen. Stellen Sie sicher, daß diese sich in dem aktuellen Verzeichnis befinden oder durch Ihre PATH- oder LIB-Umgebungsvariablen verfügbar sind. (Weitere Informationen zum Setzen von Umgebungsvariablen finden Sie in Abschnitt 1.7.)

Programm	***Aufgabe***
BC.EXE	Erstellt Objektdateien (Dateien mit der Erweiterung .OBJ) von Ihrem Programm, die gebunden werden können.
LINK.EXE	Bindet die von BC erstellten Objektdateien.
BRUN40.LIB	Bietet Unterstützungsroutinen, die es Ihrem BASIC-Programm ermöglichen, solche Aufgaben wie Benutzereingaben oder Ausgabe von Text auf den Bildschirm auszuführen. Die Routinen werden von LINK.EXE mit Ihrem Programm gebunden. Die Bibliothek BRUN40.LIB wird als Laufzeitmodul-Bibliothek bezeichnet. Jedes mit dieser Bibliothek gebundene Programm erfordert das Vorhandensein des Laufzeitmoduls BRUN40.EXE, um korrekt zu laufen.
BCOM40.LIB	Tut dasselbe wie BRUN40.LIB, mit der Ausnahme, daß mit dieser Bibliothek erstellte ausführbare Dateien nicht die Unterstützung des Laufzeitmoduls BRUN40.EXE erfordern. BCOM40.LIB wird als die alternative Laufzeitmodul-Bibliothek bezeichnet.

Die folgenden Schritte zeigen Ihnen, wie Sie eine ausführbare Datei von einem aktuell in QuickBASIC geladenen Programm erstellen:

1. Wählen Sie den Befehl **EXE-Datei erstellen** aus dem Menü **Ausführen**. Es erscheint das Dialogfeld **EXE-Datei erstellen** (siehe Abbildung 6.3)

Abbildung 6.3 Das Dialogfeld ***EXE-Datei erstellen***



a) Markieren Sie eine dieser beiden Optionen.

b) Geben Sie den Namen der zu erstellenden ausführbaren Datei ein.

c) Wählen Sie dieses Kontrollfeld zur Überprüfung von Laufzeitfehlern.

2. Geben Sie einen anderen Basisnamen in das Textfeld ein, wenn Sie Ihre Datei umbenennen möchten; andernfalls lassen Sie das Textfeld unberücksichtigt.
3. Wählen Sie entweder die Option "EXE benötigt BRUN40.EXE" oder die Option "Selbständige EXE-Datei" (deren jeweilige Vorteile sind ausführlich in den Abschnitten 6.2.2.1 und 6.2.2.2 beschrieben):

   - Die Option "EXE benötigt BRUN40.EXE" produziert eine kleine ausführbare Datei, die jedoch nur in Kombination mit dem Laufzeitmodul BRUN40.EXE gestartet werden kann.
   - Die Option "Selbständige EXE-Datei" produziert eine Datei, die ohne BRUN40.EXE läuft.

4. Schalten Sie die Kontrollfläche "Debug-Code erstellen" ein, wenn Sie wünschen, daß Ihre ausführbare Datei Code enthält, der Meldungen zur Fehlerbehandlung erzeugt und während der Laufzeit Fehlerpositionen mitteilt, oder wenn Sie wünschen, daß Ihre ausführbare Datei auf STRG+UNTBR reagiert. Beachten Sie, daß diese Option keine CodeView-kompatiblen ausführbaren Dateien produziert und darüber hinaus zu einer größeren und etwas langsameren ausführbaren Datei führt. Erklärungen zu Fehlern, die mit dieser Option behandelt werden, finden Sie in Abschnitt 6.2.3, "Prüfung auf Laufzeitfehler in ausführbaren Dateien".
5. Wählen Sie entweder die Befehlsfläche "EXE erstellen" oder die Befehlsfläche "EXE erstellen und beenden":
   - Die Befehlsfläche "EXE erstellen" erstellt eine ausführbare Datei und kehrt anschließend zu QuickBASIC zurück.
   - Die Befehlsfläche "EXE erstellen und beenden" erstellt eine ausführbare Datei und bringt Sie anschließend zur DOS-Anfrage zurück.

6.2.1 Quick-Bibliotheken und ausführbare Dateien

Wenn Sie die Befehlszeilen-Option **/l** verwendet haben, um beim Start von QuickBASIC eine Quick-Bibliothek zu laden, sucht der Compiler nach der selbständigen (.LIB) Bibliothek, die mit der genannten Quick-Bibliothek übereinstimmt, wenn der Compiler eine ausführbare Datei erstellt.

Quick-Bibliotheken bieten schnellen Zugriff auf Ihre selbsterstellten Prozedur-Bibliotheken, Sie können diese jedoch nur innerhalb der QuickBASIC-Umgebung verwenden. Der Compiler muß Zugriff auf die korrespondierende selbständige (.LIB) Bibliothek haben, um die Prozeduren der Quick-Bibliothek in eine ausführbare Datei einzufügen. Es ist daher ratsam, beide Arten von Bibliotheken - Quick-Bibliotheken und .LIB-Bibliotheken - in dasselbe Verzeichnis zu schreiben. (Der Befehl **Bibliothek erstellen** aus dem Menü **Ausführen** erstellt beide Arten von Bibliotheken automatisch. Weitere Informationen zur Erstellung und Verwendung von Quick-Bibliotheken finden Sie in Kapitel 8, "Quick-Bibliotheken", und eine Erläuterung von selbständigen Bibliotheken finden Sie in Kapitel 9, "Wie Sie aus DOS heraus kompilieren und binden".)

6.2.2 Betrachtungen zum Laufzeitmodul

Sie können, abhängig davon, ob Ihr ausführbares Programm Zugriff auf das Laufzeitmodul BRUN40.EXE erfordert oder nicht, zwei verschiedene Arten von ausführbaren Dateien erstellen.

6.2.2.1 Wie Sie das Laufzeitmodul einsetzen

Programme, die mit der Option "EXE benötigt BRUN40.EXE" verarbeitet sind, benötigen während der Laufzeit Zugriff auf das Laufzeitmodul BRUN40.EXE. Das Laufzeitmodul enthält Code, der zur Implementierung der Sprache BASIC benötigt wird. Die Verwendung des Laufzeitmoduls bietet die folgenden Vorteile:

- Die ausführbare Datei ist viel kleiner. Falls mehrere Programme auf der Diskette abgelegt sind, wird merklich Platz gespart.
- **COMMON**-Variablen und geöffnete Dateien bleiben zwischen **CHAIN**-Anweisungen erhalten. Dies kann nützlich sein in Systemen von Programmen, die Daten gemeinsam benutzen. In selbständigen Programmen bleiben **COMMON**-Variablen und geöffnete Dateien zwischen **CHAIN**-Anweisungen nicht erhalten.
- Das Laufzeitmodul BRUN40.EXE verbleibt im Speicher, so daß es nicht für jedes Programm in einem System verketteter Programme neu geladen werden muß.

6.2.2.2 Wie Sie selbständige Programme erstellen

Wenn Sie die Option "Selbständige EXE-Datei" verwenden, erfordert die ausführbare Datei keinen Zugriff auf das Laufzeitmodul BRUN40.EXE. Da Dateien, die mit dieser Option erstellt wurden, jedoch die in BRUN40.EXE gefundenen Unterstützungsroutinen enthalten, erfordern diese Dateien mehr Diskettenplatz als Dateien, die mit der Option "EXE benötigt BRUN40.EXE" erstellt wurden. Darüber hinaus konservieren die mit dieser Option erstellten Dateien keine Variablen, die in **COMMON**-Anweisungen aufgeführt sind, wenn eine **CHAIN**-Anweisung die Kontrolle an ein anderes Programm überträgt.

Dateien, die mit dieser Option erstellt werden, haben die folgenden Vorteile:

- Sie können während der Laufzeit RAM-Speicherplatz sparen, wenn Sie kleine einfache Programme haben, die nicht alle Routinen des Laufzeitmoduls benötigen.
- Die Ausführung des Programms erfordert es nicht, daß sich das Laufzeitmodul zur Ausführungszeit auf der Diskette befindet. Dies ist wichtig, wenn Sie Programme schreiben, die von Benutzern kopiert werden, da ein unerfahrener Benutzer vielleicht nicht weiß, daß auch das Laufzeitmodul kopiert werden muß.

6.2.3 Prüfung auf Laufzeitfehler in ausführbaren Dateien

Wenn Sie das Kontrollfeld "Debug-Code erstellen" in dem Dialogfeld **EXE-Datei erstellen** einschalten, während Sie eine ausführbare Datei erstellen, werden die folgenden Bedingungen während der Programmausführung geprüft:

- Arithmetischer Überlauf. Alle arithmetischen Operationen, sowohl ganzzahlige als auch Gleitkomma-Operationen, werden auf Überlauf und Unterlauf geprüft.
- Datenfeldgrenzen. Datenfeldindizes werden darauf überprüft, ob sie innerhalb der in den **DIM**-Anweisungen angegebenen Grenzen liegen.
- Zeilenpositionen. Der erzeugte Binärcode enthält zusätzliche Tabellen, so daß die Laufzeitfehlermeldungen die Zeilen kennzeichnen können, in denen Fehler auftreten.
- **RETURN**-Anweisungen. Jede **RETURN**-Anweisung wird auf eine vorhergehende **GOSUB**-Anweisung überprüft.
- STRG+UNTBR, die Tastenkombination zur Beendigung der Programmausführung. Nach der Ausführung jeder Zeile überprüft das Programm, ob der Benutzer STRG+UNTBR betätigt hat. Trifft dies zu, hält das Programm an.

  Beachten Sie, daß Sie ein Programm mit eingeschaltetem Kontrollfeld "Debug-Code erstellen" kompilieren müssen, wenn Sie wünschen, daß es auf STRG+UNTBR reagiert. Andernfalls hält ein Programm nur unter einer der folgenden beiden Bedingungen an:

  1. Wenn ein Benutzer Daten als Antwort auf die Anfrage einer **INPUT**-Anweisung eingibt.
  2. Wenn der Programmcode ausdrücklich auf STRG+UNTBR prüft.

Wenn Sie die Option "Debug-Code erstellen" nicht verwenden, dann erzeugen Fehler bei Datenfeldgrenzen, **RETURN** ohne **GOSUB**-Fehler und Fehler mit arithmetischem Überlauf keine Fehlermeldungen. Die Programmergebnisse sind möglicherweise unvorhersehbar.

6.2.4 Gleitkomma-Arithmetik in ausführbaren Dateien

Wenn Sie ein Programm in eine ausführbare Datei kompilieren, führt die ausführbare Datei Gleitkommaberechnungen schneller und effizienter durch, als wenn dasselbe Programm in der QuickBASIC-Umgebung läuft. Dies liegt daran, daß ausführbare Dateien auf Geschwindigkeit und Genauigkeit optimiert werden, indem die Reihenfolge, mit der sie bestimmte arithmetische Operationen vornehmen, verändert wird. Ausführbare Dateien nutzen darüber hinaus einen numerischen Koprozessor (bzw. die Emulation einer Koprozessorfunktion) besser aus als deren Gegenstücke, die innerhalb von QuickBASIC laufen.

Ein Nebeneffekt dieser zusätzlichen Genauigkeit ist, daß Sie vielleicht einen Unterschied in der Art bemerken, mit der ein kompiliertes Programm Vergleichsoperatoren wie zum Beispiel < oder = einsetzt, um Werte einfacher oder doppelter Genauigkeit zu vergleichen. Zum Beispiel gibt der folgende Code-Ausschnitt das Wort `Gleich` aus, wenn er in der QuickBASIC-Umgebung ausgeführt wird, gibt aber das Wort `Ungleich` aus, wenn er als ausführbare Datei ausgeführt wird:

```
B!=1.0
A!=B!/3.0
.
.
.
IF A!=B!/3.0 THEN PRINT "Gleich" ELSE PRINT "Ungleich"
```

Dieses Problem hat seine Ursache in der Durchführung der Berechnung `B!/3.0` innerhalb eines Vergleichs. Das kompilierte Programm speichert das Ergebnis dieser Berechnung in dem mathematischen Koprozessor, der diesem Ergebnis einen höheren Genauigkeitsgrad gibt, als der in der Variablen `A!` gespeicherte Wert, und somit die Ungleichheit verursacht.

Sie können solche Probleme mit vergleichenden Ausdrücken vermeiden, indem alle Berechnungen außerhalb der Vergleiche durchgeführt werden. Der folgende umgeschriebene Code-Ausschnitt ergibt dasselbe Ergebnis, wenn er in der Umgebung sowie als ausführbare Datei läuft:

```
B!=1.0
A!=B!/3.0
.
.
.
Tmp!=B!/3.0
IF A!=Tmp! THEN PRINT "Gleich" ELSE PRINT "Ungleich"
```

Weitere Informationen zu Vergleichen, die Gleitkommawerte betreffen, finden Sie in Kapitel 3, "Ausdrücke und Operatoren", im *BASIC-Befehlsverzeichnis*.

6.3 Wie Sie SUB- und FUNCTION-Prozeduren in Ihren Programmen einsetzen

QuickBASIC unterstützt zwei Typen untergeordneter Prozeduren: Solche, die innerhalb von **SUB...END SUB**-Anweisungen definiert sind und solche, die innerhalb von **FUNCTION...END FUNCTION**-Anweisungen definiert sind. Dieser Abschnitt beschreibt, wie Sie **SUB**- und **FUNCTION**-Prozeduren erstellen. (Weitere Informationen zur Verwendung von Prozeduren finden Sie in Kapitel 2, "Prozeduren: Unterprogramme und Funktionen", in *Programmieren in BASIC: Ausgewählte Themen*.)

Hinweis **SUB**- und **FUNCTION**-Prozeduren sind nicht länger in Include-Dateien erlaubt, wie das in früheren QuickBASIC-Versionen der Fall war. Weitere Informationen zur Anpassung von Prozeduren, die ursprünglich in Include-Dateien verwendet wurden, für den Einsatz in QuickBASIC-Modulen finden Sie in Anhang B, "Unterschiede zu früheren QuickBASIC-Versionen".

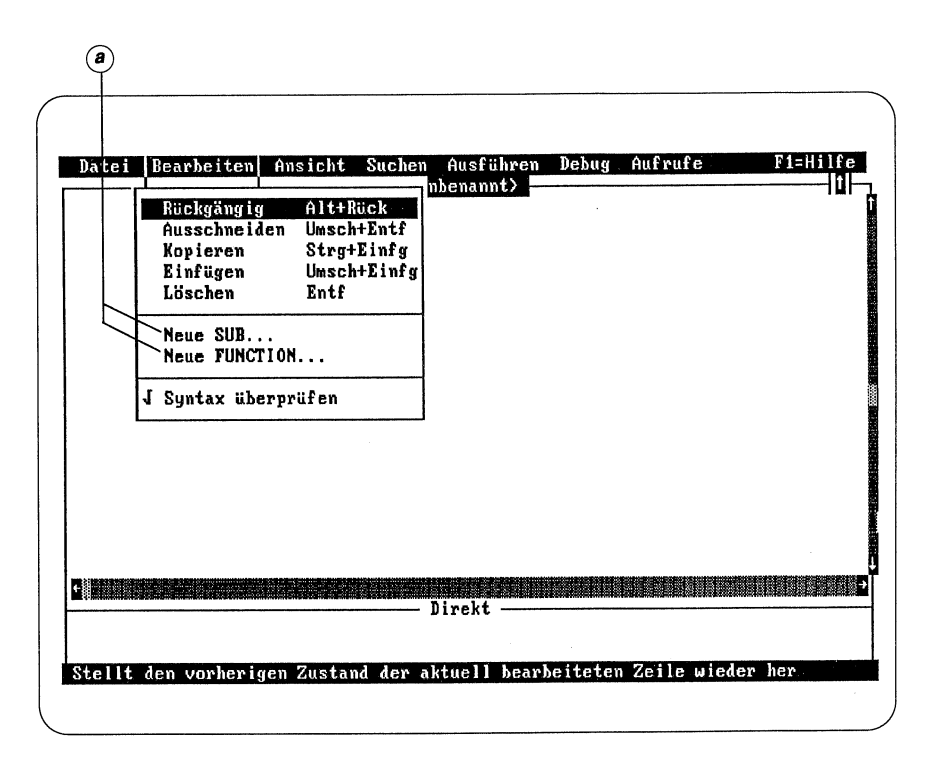
6.3.1 Wie Sie SUB- oder FUNCTION-Prozeduren erstellen (Neue SUB, Neue FUNCTION)

Wenn Sie eine Prozedur definieren, wird diese Teil des Moduls im aktiven Arbeitsbereich. Wenn Sie dieses Modul als Datei abspeichern, wird jede Prozedur, die diese Datei enthält, Teil der Datei. Prozeduren können im Hauptmodul oder in anderen Modulen definiert sein. Wenn Sie sich nach der Erstellung einer Prozedur dazu entschließen, diese an anderer Stelle zu plazieren, können Sie die Prozedur zwischen Modulen bewegen, indem Sie den Befehl **SUBs** aus dem Menü **Ansicht** verwenden. Eine Prozedur darf jedoch nur in einem Modul eines Programms, das diese Prozedur aufruft, erscheinen.

Prozeduren haben eigene Befehle in dem Menü **Bearbeiten**, die in Abbildung 6.4 gezeigt sind. Folgen Sie diesen Schritten, um eine **SUB**- oder **FUNCTION**-Prozedur zu erstellen:

1. Wählen Sie den Befehl **Neue SUB** oder **Neue FUNCTION** aus dem Menü **Bearbeiten**. Es erscheint das in Abbildung 6.4 gezeigte Dialogfeld.

*Abbildung 6.4 Das Menü **Bearbeiten***



*a) Wählen Sie diese Befehle, um eine neue **SUB** oder **FUNCTION** anzulegen.*

2. Geben Sie in das Textfeld einen Namen für die Prozedur ein.
3. Betätigen Sie die EINGABETASTE.

   QuickBASIC öffnet ein Fenster und beginnt die Prozedur mit den Anweisungen **FUNCTION...END FUNCTION** oder **SUB...END SUB** und dem Namen, den Sie in das Textfeld eingegeben haben.

Wenn Sie eine Prozedur beginnen, identifiziert die Titelleiste den Inhalt des Arbeitsbereiches mit dem Namen des Hauptmoduls, gefolgt von dem Namen Ihrer Prozedur.

Für das Erstellen von Prozeduren gibt es eine Tastenkurzkombination. Geben Sie einfach eines der Schlüsselwörter **SUB** oder **FUNCTION**, gefolgt von dem Namen der Prozedur, ein. Wenn Sie die EINGABETASTE betätigen, führt QuickBASIC alle vorhergehenden Schritte automatisch durch.

Wichtig Wenn ein Programm geladen wird, das auf Modul-Ebene eine **DEF***Typ*-Anweisung (anders als die Anweisung **DEFSNG**, der voreingestellte BASIC-Datentyp) enthält, fügt QuickBASIC sofort eine Kopie Ihrer Anweisung über der **SUB**- oder **FUNCTION**-Anweisung in jeder Prozedur ein, die innerhalb dieses Moduls erstellt ist. Wenn Sie Ihre Meinung ändern und wünschen, daß ein anderer Datentyp überwiegt, ersetzen Sie die vorhandene **DEF***Typ*-Anweisung einfach durch diejenige, die Sie wünschen.

In einem Prozedurfenster sind oberhalb der **SUB**- oder **FUNCTION**-Anweisung ausführbare Anweisungen und leere Zeilen nicht erlaubt. Wenn Sie eine leere Zeile erstellen, um auf dieser eine **DEF***Typ*-Anweisung (welche erlaubt ist) einzugeben, gibt QuickBASIC daher eine Fehlermeldung aus und fragt Sie, ob eine Anmerkung korrekt ist. Betätigen Sie die EINGABETASTE; QuickBASIC plaziert dann ein Kommentarzeichen am Beginn der Zeile. Sie können anschließend das Kommentarzeichen löschen oder überschreiben, um Ihre **DEF***Typ*-Anweisung einzugeben.

Wenn über der Prozedurdefinition keine **DEF***Typ*-Anweisung erscheint, dann hat die Prozedur einfache Genauigkeit als Standard-Datentyp, weil dieser der Standard-Datentyp in BASIC ist.

Wenn Sie ein Modul erstellen und vergessen, eine **DEF***Typ*-Anweisung auf Modul-Ebene zu schreiben, hat jede Prozedur, die Sie innerhalb des Moduls erstellen, einfache Genauigkeit als ihren angenommenen Datentyp, obwohl kein ausdrücklicher Hinweis darauf über der Prozedurdefinition vorhanden ist. Wenn Sie später zurückgehen und eine **DEF***Typ*-Anweisung auf Modul-Ebene setzen, wird diese nicht zu einer bereits existierenden Prozedur kopiert. Wenn Sie einen anderen voreingestellten Datentyp als den der einfachen Genauigkeit in solchen Prozeduren überwiegen lassen möchten, müssen Sie ausdrücklich eine **DEF***Typ*-Anweisung in die Prozedur eingeben. Weitere Informationen zu Datentypen finden Sie in Kapitel 2, "Datentypen", im *BASIC-Befehlsverzeichnis.*

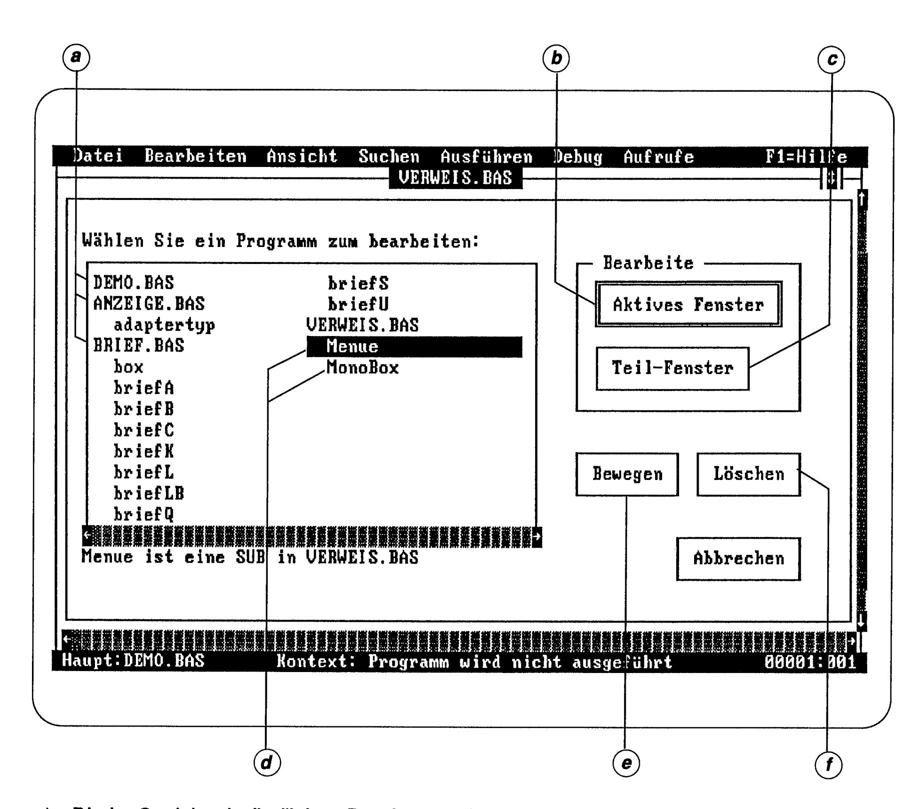
6.3.2 Prozeduren abspeichern und benennen

Wenn Sie Ihr Programm abspeichern, werden Prozeduren nicht als unabhängige Diskettendateien, so wie es bei Modulen der Fall ist, abgespeichert, so daß Prozedurnamen mit den Benennungsvereinbarungen von QuickBASIC und nicht mit denen von DOS verwaltet werden. (Ein Prozedurname darf jeder gültige BASIC-Name mit bis zu 40 Zeichen Länge sein. Die Regeln für die Benennung von Prozeduren sind dieselben wie die für die Benennung von Variablen. Weitere Informationen zu Variablennamen in BASIC finden Sie in Kapitel 2, "Datentypen", im *BASIC-Befehlsverzeichnis.*)

6.3.3 Wie Sie Module und Prozeduren betrachten und bearbeiten (SUBs)

Sie können eine Liste der Module und Prozeduren des aktuellen Programmes sehen, indem Sie den Befehl **SUBs** aus dem Menü **Ansicht** wählen (oder F2 betätigen). Abbildung 6.5 zeigt das Dialogfeld **SUBs**.

*Abbildung 6.5 Das Dialogfeld **SUBs***



a) Die im Speicher befindlichen Dateien werden in Großbuchstaben aufgeführt.

b) Die markierte Position (Modul/Prozedur) wird in das aktive Fenster gesetzt.

c) Das aktive Fenster wird beibehalten, die markierte Position (Modul/Prozedur) wird in das neue Fenster gesetzt.

d) Prozeduren stehen eingerückt unter den Modulen, die die Prozeduren enthalten.

e) Prozeduren werden zwischen Modulen bewegt.

f) Eine Datei wird entfernt, oder eine Prozedur wird komplett aus einem Modul gelöscht.

Die Prozedurnamen werden in dem Listenfeld eingerückt unter den Modulen angezeigt, in denen die Prozeduren definiert sind. Um eine **SUB**- oder **FUNCTION**-Prozedur zu bearbeiten, bewegen Sie diese wie folgt in einen Arbeitsbereich:

1. Positionieren Sie die Hervorhebung auf den Namen der Prozedur.
2. Wählen Sie entweder die Befehlsfläche "Aktives Fenster" oder "Teil-Fenster":
   - "Aktives Fenster" ersetzt das aktuell aktive Fenster durch den Text der aus der Liste gewählten Position (Modul/Prozedur).
   - "Teil-Fenster" schreibt die aus der Liste gewählte Position (Modul/Prozedur) in ein zweites Fenster, wenn nur ein Fenster geöffnet ist. Wenn bereits zwei Fenster geöffnet sind, ersetzt die neue Position (Modul/Prozedur) das Fenster, das nicht aktiv ist.

Wenn ein Modul mehrere Prozeduren enthält, können Sie diese in alphabetischer Reihenfolge durchlaufen, indem Sie wiederholt den Befehl **Nächste SUB** aus dem Menü **Ansicht** wählen (oder indem Sie die Tastenkurzkombination UMSCHALTTASTE+F2 verwenden, um sich nach unten zu bewegen, bzw. STRG+F2, um sich nach oben zu bewegen).

6.3.4 Automatische Prozedurdeklarationen

Wenn Sie ein Programm als Diskettendatei speichern (oder, falls es sich um ein mehrmoduliges Programm handelt, als eine Sammlung von Diskettendateien), erstellt QuickBASIC für jede Prozedur in einem Programm eine Deklaration. Eine Deklaration besteht aus dem Schlüsselwort **DECLARE**, gefolgt entweder von dem Schlüsselwort **SUB** oder dem Schlüsselwort **FUNCTION**, und daran anschließend dem Namen der Prozedur sowie einer Liste der formalen Prozedurparameter.

Weitere Informationen zur Bedeutung von Deklarationen, und wie diese in Programmen eingesetzt werden, finden Sie in Kapitel 2, "Prozeduren: Unterprogramme und Funktionen", in *Programmieren in BASIC: Ausgewählte Themen* sowie in Kapitel 4, "Programme und Module", im *BASIC-Befehlsverzeichnis.*

Beispiel

Im nächsten Beispiel erzeugt QuickBASIC die erste Anweisung, wenn Sie das Programm mit dem Befehl **Speichern unter** aus dem Menü **Datei** speichern.

```
DECLARE SUB DemoDekl (Erst$, Zweit%)   'Diese
                                       'Deklaration auf
CALL DemoDekl ("Du hier", 2)           'Modul-Ebene ist
                                       'ein Ergebnis
SUB DemoDekl (Erst$, Zweit%) STATIC    'des Aufrufs
   PRINT Erst$ " hat " Zweit% "Wörter" 'dieser Proze-
END SUB                                'durdefinition.
```

6.4 Wie Sie Module in einem Programm zusammenfassen

Ein Modul ist eine einzelne Diskettendatei, die BASIC-Anweisungen enthält. Ein Modul kann ein abgeschlossenes Programm sein oder eine Datei, die eine oder mehrere **SUB**- oder **FUNCTION**-Prozeduren enthält, die von anderen Modulen aufgerufen werden.

Mit traditionellen Compiler/Linker-Programm-Entwicklungssystemen besteht die einzige Möglichkeit, ein mehrmoduliges Programm zu erzeugen, darin, daß jedes Modul als separate Objektdatei kompiliert wird, und diese Objektdateien anschließend gebunden werden, um eine ausführbare Datei auf Diskette zu bilden. In QuickBASIC jedoch können Sie mehrmodulige Programme im *Speicher* erstellen und diese genauso ausführen, wie Sie es mit einem einmoduligen Programm tun würden. Zum Beispiel kann ein neues Programm mehrere vorhandene Module verwenden, wobei jedes dieser Module mehrere Prozeduren enthalten kann, die Sie in Ihrem Programm verwenden möchten.

Die folgenden Schritte zeigen, wie Prozeduren aus separaten Modulen in einem neuen Programm zusammengefaßt werden:

1. Starten Sie QuickBASIC, und geben Sie als Argument der Befehlszeile den Namen ein, den Sie Ihrem Programm geben möchten. Geben Sie zum Beispiel

   ```
   qb programm
   ```

   ein. Dies wird der Name Ihres Hauptmoduls sein.
2. Verwenden Sie den Befehl **Datei laden** aus dem Menü **Datei**, um jedes Modul zu laden, das Prozeduren enthält, die in dem neuen Programm aufgerufen werden.
3. Schreiben Sie den Code Ihres Hauptmoduls. Sie können Prozeduren jedes aktuell geladenen Modules aufrufen.
4. Stellen Sie sicher, daß sich jede von dem Programm benötigte Include-Datei in dem aktuellen Arbeitsverzeichnis befindet, oder daß der Metabefehl **$INCLUDE** entweder einen vollständigen oder relativen Pfad zu der Include-Datei anlegt, wie in dem folgenden Beispiel:

   ```
   ' $INCLUDE: '..\include\grafik.bi'
   ' $INCLUDE: 'c:\include\grafik.bi'
   ```

5. Wählen Sie **Start** aus dem Menü **Ausführen.**

   Stellen Sie sicher, daß Ihr Programm die Ergebnisse liefert, die Sie wünschen.

6. Wählen Sie **Alles speichern** aus dem Menü **Datei**, um alles als ein einziges Programm abzuspeichern.

In Wirklichkeit sind zahlreiche Zwischenschritte notwendig, bis mehrere Module als ein Programm zusammenarbeiten. Wenn zum Beispiel eines der Nicht-Hauptmodule eine **FUNCTION**-Prozedur enthält, erzeugt QuickBASIC die **DECLARE**-Anweisung innerhalb dieses Moduls. Bevor das Modul korrekt arbeitet, müssen Sie jedoch die **DECLARE**-Anweisung in jedes Modul kopieren, in dem die Prozedur aufgerufen wird. Eine bequeme Möglichkeit, dies durchzuführen, besteht darin, alle **DECLARE**-Anweisungen in eine Include-Datei zu kopieren und anschließend den Metabefehl **$INCLUDE** zu verwenden, um die Anweisungen in die Module einzufügen, die diese benötigen. Die Sprachregeln zur Verwaltung mehrmoduliger Programme finden Sie in Kapitel 4, "Programme und Module", im *BASIC-Befehlsverzeichnis*. Die Regeln zur Verwaltung von Prozeduren finden Sie in Kapitel 2, "Prozeduren: Unterprogramme und Funktionen", in *Programmieren in BASIC: Ausgewählte Themen*".

7 Debuggen während der Programmierung

Kapitel 5, "Bearbeiten", und Kapitel 6, "Wie Sie QuickBASIC-Programme erstellen und ausführen", haben Ihnen gezeigt, wie QuickBASIC Syntaxfehler auf einer einzelnen Zeile während der Eingabe Ihres Programmes entdeckt. Dieses Kapitel zeigt, wie QuickBASIC Ihnen dabei hilft, Fehler ausfindig zu machen, die sich aus Fehlern in der Programmlogik ergeben.

Wenn Sie dieses Kapitel gelesen haben, werden Sie wissen, wie die folgenden Debug-Aufgaben durchgeführt werden:

- QuickBASIC einsetzen, um Fehler während der Eingabe Ihres Programmes zu vermeiden.
- Fehler mit QuickBASIC isolieren und korrigieren, um
  - Die Programmausführung Anweisung für Anweisung zu verfolgen.
  - Ihr Programm solange auszuführen, bis es eine bestimmte Zeile oder Zeilen erreicht hat.
  - Variablenwerte während der Programmausführung zu verfolgen.
  - Die Ausführung zeitweise zu unterbrechen, wenn eine Variable bzw. ein Ausdruck einen festgelegten Wert oder eine festgelegte Bedingung annimmt.
  - Neue Werte, Ausdrücke und ganze Code-Abschnitte innerhalb eines unterbrochenen Programmes zu ersetzen und anschließend das Programm fortzusetzen.
  - Die letzten 20 ausgeführten Zeilen zu wiederholen.

7.1 Wie Sie mit QuickBASIC debuggen

Früher oder später muß sich jeder Programmierer mit dem Debuggen – also dem Prozeß des Lösens von Problemen, die Ihr Programm entweder davon abhalten, überhaupt zu laufen, oder es veranlassen, unvorhersehbare Ergebnisse zu produzieren – beschäftigen. Unabhängig davon, wie gut Sie als Programmierer sind, besteht die Möglichkeit, daß Sie ein Programm, das – auf dem Papier – so aussieht, als müßte es sofort richtig laufen, debuggen können.

Die Debug-Techniken, die Sie bisher für Ihre BASIC-Programme verwendet haben, können von der Beobachtung des Programmablaufs und vorausschauenden Vermutungen bis zum eigentlichen Einsetzen der Debug-Werkzeuge reichen, die mit der jeweiligen BASIC-Version zur Verfügung stehen. Wenn Sie in BASICA programmiert haben, sind Ihre Hilfsmittel die Anweisungen **STOP**, **PRINT**, **CONT**, **TRON** und **TROFF**.

Wenn Sie mit einer kompilierenden Version von BASIC programmiert haben, haben Sie Ihre kompilierten Programme eventuell mit einem Hilfsprogramm, das den Namen "symbolischer Debugger" trägt, debugged. Dieser Debugger erlaubt es Ihnen, den Fluß Ihrer Programmlogik zu überwachen, während Sie gleichzeitig beobachten können, was das Programm tut.

Beide dieser Hilfsmittel sind begrenzt. In BASICA gibt es keine Möglichkeit, eine Variable oder einen Ausdruck während der Programmausführung anzuzeigen, und jede Änderung an einem unterbrochenen Programm zwingt Sie, den gesamten Vorgang neu zu starten.

Symbolische Debugger sind zwar leistungsfähige Hilfsprogramme, bevor Sie aber einen symbolischen Debugger einsetzen können, müssen Sie Ihr Programm erfolgreich kompilieren. Dieser Kompilier-Prozeß kann viele Wiederholungen der folgenden Schleife nach sich ziehen:

- Versuchen, Ihr Programm zu kompilieren.
- Erhalten eines Fehlers, gelegentlich begleitet von verschlüsselten Kompilier-Meldungen.
- Entziffern der Fehlermeldung, anschließende Editierung Ihrer Quelldatei.
- Erneutes Kompilieren.

Alle oben aufgeführten Punkte bringen Sie lediglich zu dem Punkt, ab dem Sie den symbolischen Debugger einsetzen können. Jedesmal, wenn Sie mit dem Debugger ein Problem einkreisen, müssen Sie die oben beschriebene Schleife abermals durchlaufen.

Wie Sie sehen werden, vereinigt QuickBASIC die Leichtigkeit des Debuggens in BASICA mit den Eigenschaften eines symbolischen Debuggers, wie Haltepunkte, Stoppbedingungen und Anzeigeausdrücke. Um Ihr Programm zu debuggen, ist es nicht notwendig, ein separates Hilfsmittel zu laden oder QuickBASIC mit einer besonderen Option aufzurufen. Wenn Sie einen Fehler bemerken, können Sie sofort beginnen, diesen zu verfolgen. Die QuickBASIC-Umgebung bietet alles, was Sie hierzu benötigen.

Zusätzlich müssen Sie Ihr Programm mit QuickBASIC nicht neu kompilieren oder binden, wenn Sie es verändern. Meistens können Sie Ihr Programm bis zu dem Punkt ausführen, an dem das Problem auftritt, die Ausführung unterbrechen, das Problem lösen und anschließend die Ausführung des Programmes an dem Punkt fortsetzen, an dem Sie es angehalten haben. In QuickBASIC debuggen Sie während der Programmierung.

7.2 Wie Sie Fehlern vorbeugen

Die beste Möglichkeit, ein fehlerfreies Programm zu bekommen, besteht darin, Fehlern vorzubeugen, bevor diese passieren. Die folgende Liste beschreibt drei Arten, wie Sie Fehlern vorbeugen können:

1. Entwerfen Sie Ihr Programm sorgfältig, noch bevor Sie mit der Eingabe beginnen.

   Eine wichtige Regel für einen guten Programmentwurf lautet, verschiedene Aufgaben, die Ihr Programm ausführen soll, zu isolieren. Solche Aufgaben können das Sortieren eines Datenfeldes oder das Separieren einer Zeile (einzelne Wörter nach Entfernung von Interpunktions- und Leerzeichen erhalten) sein. Anschließend können Sie **SUB**- oder **FUNCTION**-Prozeduren schreiben und debuggen, die solche Aufgaben durchführen. Eine kleine Prozedur ist viel leichter zu debuggen als ein großes Programm, das keine Prozeduren verwendet.

   Ein zusätzlicher Vorteil von Prozeduren liegt darin, daß Sie diese in einer separaten Datei, die als Modul bezeichnet wird, speichern und die Prozeduren anschließend in anderen Programmen einsetzen können, die dieselben Aufgaben ausführen. (Weitere Informationen zu Prozeduren und Modulen finden Sie in Kapitel 2, "Prozeduren: Unterprogramme und Funktionen", in *Programmieren in BASIC: Ausgewählte Themen* sowie in Kapitel 4, "Programme und Module", im *BASIC-Befehlsverzeichnis.*)

2. Verwenden Sie das Direkt-Fenster.

   Während der Programmierung können Sie das Direkt-Fenster unten auf dem QuickBASIC-Bildschirm einsetzen, um kleine Code-Abschnitte zu isolieren und zu testen. Wenn diese Abschnitte selbständig korrekt laufen, können Sie diese in den Hauptblock des Programms bewegen. Weitere Informationen zu dem Direkt-Fenster finden Sie in Abschnitt 6.1.5.

3. Starten Sie Ihren Code häufig.

   Da QuickBASIC jede Anweisung während der Eingabe überprüft und in ausführbaren Code übersetzt, können Sie jeden neuen Code-Abschnitt immer im Zusammenhang mit dem restlichen Programm testen. Dies hilft Ihnen, einfache Fehler zu finden, die in einem fertigen Programm schwieriger aufgespürt werden könnten.

   Wenn Sie zum Beispiel die Schleife in dem folgenden Ausschnitt isolierten, würden Sie wahrscheinlich sehr schnell den Fehler bemerken (die Variable `I%` wird nicht hochgezählt, da das `I` sich auf eine ganz andere Variable bezieht).

```
DIM DatFeld$ (1 TO 20)
.
.
.
I% = 1
DO
   INPUT Temp$
   IF Temp$ <> "" THEN
      DatFeld$(I%) = Temp$
      I% = I + 1
   END IF
LOOP UNTIL Temp$ = "" OR I% > 20
```

Eine Möglichkeit, dies zu testen, besteht darin, das Datenfeld mit einigen Zeichenkettenausdrücken zu laden und anschließend folgendes in das Direkt-Fenster einzugeben:

```
FOR I% = 1 TO 20: PRINT Datfeld$(I%): NEXT
```

Die Ausgabe wird ein nützlicher Hinweis sein, daß etwas korrigiert werden muß.

Eine andere Technik besteht darin, einen "Anzeigeausdruck" zu setzen, das heißt, Sie können den Wert der Variablen `I%` beobachten, während Sie die Schleife verfolgen. Der Abschnitt 7.3.4.4 zeigt Ihnen, wie ein Ausdruck während einer Programmausführung beobachtet werden kann.

7.3 Die Debug-Eigenschaften von QuickBASIC

Dieser Abschnitt beschreibt die vollintegrierten Debug-Eigenschaften von QuickBASIC. Der Abschnitt 7.3.1 definiert einige Schlüsselbegriffe und Konzepte des Debuggens. Der Abschnitt 7.3.2 stellt das Menü **Debug** vor. Die Abschnitte 7.3.3 bis 7.3.5 erläutern, wie Sie die Debug-Befehle von QuickBASIC einsetzen.

Während Sie diese Abschnitte lesen, sollten Sie im Gedächtnis behalten, daß diese Eigenschaften immer verfügbar sind, wenn Sie ein Programm innerhalb der QuickBASIC-Umgebung entwickeln. Sie müssen nicht zwischen dem Editor und einem Debug-Werkzeug hin- und herschalten oder einen besonderen Debug-Modus wählen.

7.3.1 Debug-Begriffe und -Konzepte

Bevor Sie weiterlesen, sollten Sie die folgenden Debug-Begriffe und -Konzepte von QuickBASIC kennen.

- **Verfolgen**

  Verfolgen versetzt Sie in die Lage zu sehen, welche Anweisung Ihres Programmes ausgeführt wird. Dies ermöglicht es Ihnen zum Beispiel zu sehen, welcher Zweig einer **IF...THEN...ELSE**-Anweisung genommen wird. QuickBASIC bietet Befehle zur Verfolgung, mit denen Sie folgendes durchführen können:

  – Anweisung für Anweisung ausführen.

  – Die Ausführung animieren, so daß jede Anweisung hervorgehoben erscheint, wenn sie ausgeführt wird.

  – Die letzten 20 ausgeführten Anweisungen vorwärts oder rückwärts verfolgen.

- **Haltepunkt**

  Ein Haltepunkt ist eine Stelle in Ihrem Programm, an der Sie die Ausführung anhalten möchten. Mit Haltepunkten können Sie testen, welche Teile eines Programmes gerade laufen. Haltepunkte können Sie ebenfalls einsetzen, um das Programm an zentralen Stellen anzuhalten und den Wert einer Variablen zu überprüfen, so daß Sie exakt sehen können, was passiert ist.

- **Stoppbedingung**

  Eine Stoppbedingung ist ein Ausdruck, der ein Programm anhält, wenn der Ausdruck wahr wird (bzw. ungleich Null).

- **Anzeigeausdruck**

  Anzeigeausdrücke ermöglichen es Ihnen, die Werte von Variablen oder Ausdrücken während der Ausführung Ihres Programmes zu beobachten.

- **Debug-Fenster**

  Das Debug-Fenster ist das Fenster, das sich auf dem oberen Teil des QuickBASIC-Bildschirms öffnet, um es Ihnen zu ermöglichen, Stoppbedingungen oder Variablenwerte während der Programmausführung zu verfolgen. Weitere Informationen zu dem Debug-Fenster finden Sie in Abschnitt 7.3.4.1.

- **Kontext**

  Der Begriff "Kontext" bezieht sich auf den Teil Ihres Programmes, der aktuell ausgeführt wird. Der Name dieses Teils (oder die Wörter "Programm läuft nicht", falls passend) erscheint auf der Statusleiste im unteren Teil des QuickBASIC-Bildschirms nach dem Wort "Kontext".

Der Kontext einer Stoppbedingung oder eines Anzeigeausdruckes bezieht sich auf den Teil Ihres Programms, der sich im aktuellen Fenster befindet, wenn Sie den Ausdruck in das Debug-Fenster einfügen. Nehmen Sie zum Beispiel an, Ihr Programm enthält eine als `Analyse` benannte **FUNCTION**-Prozedur, und Sie möchten den Wert einer Variablen überwachen, die `Zaehler` heißt. Falls Sie in das Debug-Fenster die Variable `Zaehler` einfügen, wenn sich die Prozedur `Analyse` im aktiven Fenster befindet, werden Sie sehen können, wie sich diese Variable verändert, während Ihr Programm die Prozedur `Analyse` ausführt. Tatsächlich geht der Name `Analyse` dem Eintrag `Zaehler` im Debug-Fenster voraus, da der Debug-Kontext der Variablen `Zaehler` die Prozedur `Analyse` ist.

Der aktuelle Wert oder die aktuelle Bedingung einer Variablen oder eines Ausdrucks kann in dem Debug-Fenster nur angezeigt werden, wenn der Debug-Kontext (angezeigt in dem Debug-Fenster) dieser Größe identisch ist mit dem Kontext der Programmausführung (gezeigt auf der Statusleiste). Andernfalls wird die Meldung `Nicht anzeigbar` ausgegeben.

Hinweis Selbst wenn Ihr Programm eine **SHARED**-Anweisung verwendet, um eine Variable zwischen der Modul-Ebene und der Prozedur-Ebene gemeinsam zu benutzen, kann diese Variable nur auf der Ebene beobachtet werden, die aktiv ist, wenn Sie die Variable in das Debug-Fenster einfügen.

Während eine **DEF FN**-Funktion aktiv ist, kann nichts angezeigt werden.

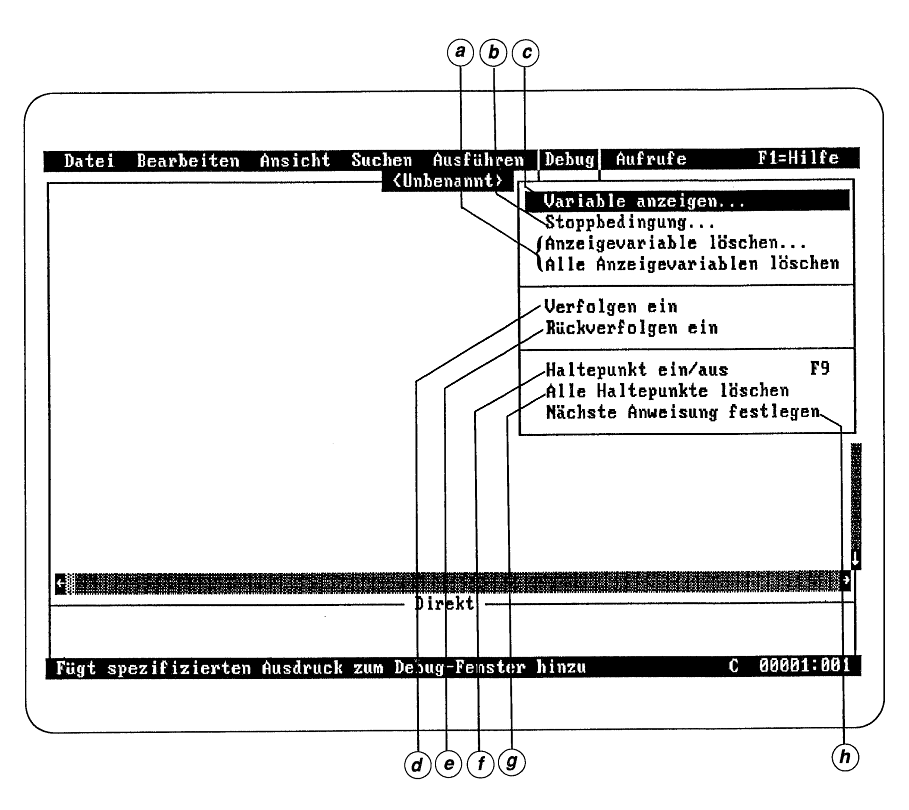
- **Direkt-Fenster**

  Das Direkt-Fenster ist das Fenster im unteren Teil des Bildschirms, in dem Sie QuickBASIC-Anweisungen direkt ausführen können. Der aktuelle Kontext der Ausführung bestimmt, auf welche Variablen eines unterbrochenen Programmes innerhalb des Direkt-Fensters zugegriffen werden kann, und welche Standard-Datentypen diese haben. Wenn zum Beispiel die nächste auszuführende Anweisung in dem unterbrochenen Programm eine **END FUNCTION**-Anweisung ist, kann auf Variablen, die lokal zu dieser Funktion sind, aus dem Direkt-Fenster zugegriffen werden. Nachdem die Anweisung ausgeführt ist, kann jedoch nicht länger auf sie zugegriffen werden. Eine Erläuterung, wie sich QuickBASIC und BASICA in der Behandlung von Standard-Datentypen unterscheiden, finden Sie in der Beschreibung zu **DEF***Typ* im *BASIC-Befehlsverzeichnis.*

7.3.2 Das Menü Debug

Obwohl die Menüs **Ansicht**, **Ausführen** und **Aufrufe** Befehle enthalten, die Sie beim Debuggen häufig verwenden, erscheinen die meisten der QuickBASIC-Debug-Befehle in dem Menü **Debug**, gezeigt in Abbildung 7.1.

Abbildung 7.1 Das Menü ***Debug***



a) Veranlaßt die teilweise oder komplette Entfernung von Positionen aus dem Debug-Fenster.

b) Es werden Namen von Variablen und Ausdrücken eingefügt, die die Programmausführung unterbrechen, wenn sie WAHR (ungleich Null) werden.

c) Es werden Namen von Variablen und Ausdrücken in dem Debug-Fenster hinzugefügt.

d) Veranlaßt animierte Ausführung, wenn das Programm läuft.

e) Schaltet die Aufzeichnung der letzten 20 ausgeführten Anweisungen ein oder aus.

f) Setzt und gibt bestimmte Haltepunkte frei.

g) Schaltet alle Haltepunkte aus.

h) Legt die nächste auszuführende Anweisung fest, wenn Sie wünschen, daß dazwischenliegende Anweisungen übersprungen werden, ohne daß diese ausgeführt werden.

7.3.3 Wie Sie eine Programmausführung verfolgen: Mehr als TRON und TROFF

Verfolgen ermöglicht es Ihnen, den Ablauf Ihres Programmes zu beobachten. QuickBASIC bietet für unterschiedliche Zwecke verschiedene Möglichkeiten zur Verfolgung. QuickBASICs Eigenschaften zur Verfolgung sind leistungsfähiger als diejenigen, die Ihnen **TRON** und **TROFF** in BASICA bieten.

7.3.3.1 Einzelschritt (F8)

Nach der Aktivierung durch die Taste F8 führt Einzelschritt eine Anweisung Ihres Programmes jedesmal aus, wenn die Taste betätigt wird. Falls die Anweisung Bildschirmausgabe erzeugt, erscheint kurz, bevor QuickBASIC in die Umgebung zurückkehrt, diese Ausgabe. (Sie können zwischen dem Ausgabebildschirm und der Umgebung hin- und herschalten, indem Sie den Befehl **Ausgabebildschirm** wählen oder F4 betätigen.)

Einzelschritt verfolgt durch jede Zeile von **SUB**- oder **FUNCTION**-Prozeduren, wenn diese in Ihrem Programm aufgerufen werden.

7.3.3.2 Prozedurschritt (F10)

Nach der Aktivierung durch die Taste F10 verhält sich Prozedurschritt wie Einzelschritt, mit der Ausnahme, daß es einen Prozeduraufruf ausführt, als wäre die gesamte Prozedur eine einzige Anweisung. Dies wird manchmal als "Übergehen" einer Prozedur bezeichnet.

Mit Prozedurschritt werden alle Anweisungen der Prozedur zwar ausgeführt, Sie können aber nur das Ergebnis des Prozeduraufrufs sehen.

7.3.3.3 Animierte Verfolgung (Verfolgen ein)

Sie können Animation einschalten, indem Sie den Befehl **Verfolgen ein** aus dem Menü **Debug** wählen. Wenn Sie nachfolgend den Befehl **Start** oder **Weiter** aus dem Menü **Ausführen** wählen, während der Befehl **Verfolgen ein** aus dem Menü **Debug** mit einem Prüfzeichen (√) gekennzeichnet ist, führt QuickBASIC Ihr Programm kontinuierlich in Zeitlupe in Einzelschritten aus. Jede Anweisung ist während der Ausführung hervorgehoben. Zusammen mit Stoppbedingungen, Haltepunkten (behandelt in Abschnitt 7.3.4.2) und der Tastenkombination STRG+UNTBR (die die Ausführung anhält), ist dies eine hilfreiche Methode festzustellen, ob der allgemeine Ablauf Ihres Programmes dem entspricht, was Sie wünschen.

Hinweis Der Umschaltbefehl **Verfolgen ein** hat keine Auswirkungen auf die Funktion der Verfolgungsbefehle F8 und F10. Diese arbeiten wie gewohnt weiter.

7.3.3.4 Rückverfolgen ein

Das Einschalten von **Verfolgen ein** veranlaßt QuickBASIC dazu, die letzten 20 von Ihrem Programm ausgeführten Anweisungen aufzuzeichnen, obwohl Sie diese Möglichkeit auch unabhängig einschalten können, indem Sie den Befehl **Rückverfolgen ein** wählen. **Rückverfolgen ein** ermöglicht es Ihnen, die letzten 20 von Ihrem Programm ausgeführten Anweisungen nach unten bzw. nach oben zu verfolgen. Dies ist sehr hilfreich, wenn Sie einen Laufzeitfehler erhalten und die Auswirkung der 20 Anweisungen sehen möchten, die dem Fehler unmittelbar vorausgehen. Zusammen mit den Tastenkombinationen zur Rückverfolgung (beschrieben in den Abschnitten 7.3.3.5 bis 7.3.3.6) ermöglicht es Ihnen dieser Schrittmodus auch, der genauen Verzweigung eines Programms durch verschachtelte Bedingungsstrukturen, wie **IF...THEN...ELSE**, zu folgen. Wenn **Rückverfolgen ein** eingeschaltet ist, ist die Ausführungsgeschwindigkeit geringer, obwohl die Ausführung nicht so wie mit **Verfolgen ein** animiert wird.

7.3.3.5 Rückverfolgung nach oben (UMSCHALTTASTE+F8)

Wenn entweder der Befehl **Verfolgen ein** oder **Rückverfolgen ein** gekennzeichnet ist, können Sie die Ausführung Ihres Programmes unterbrechen, indem Sie STRG+UNTBR und anschließend UMSCHALTTASTE+F8 betätigen, um die letzten 20 ausgeführten Anweisungen nach oben zu verfolgen.

7.3.3.6 Rückverfolgung nach unten (UMSCHALTTASTE+F10)

Nachdem Sie mit den Tasten Rückverfolgung nach oben Anweisungen nach oben durchgesehen haben, können Sie dieselbe Folge von Anweisungen nach unten durchlaufen, indem Sie UMSCHALTTASTE+F10 einmal für jede Anweisung betätigen.

7.3.4 Haltepunkte, Stoppbedingungen und Anzeigeausdrücke

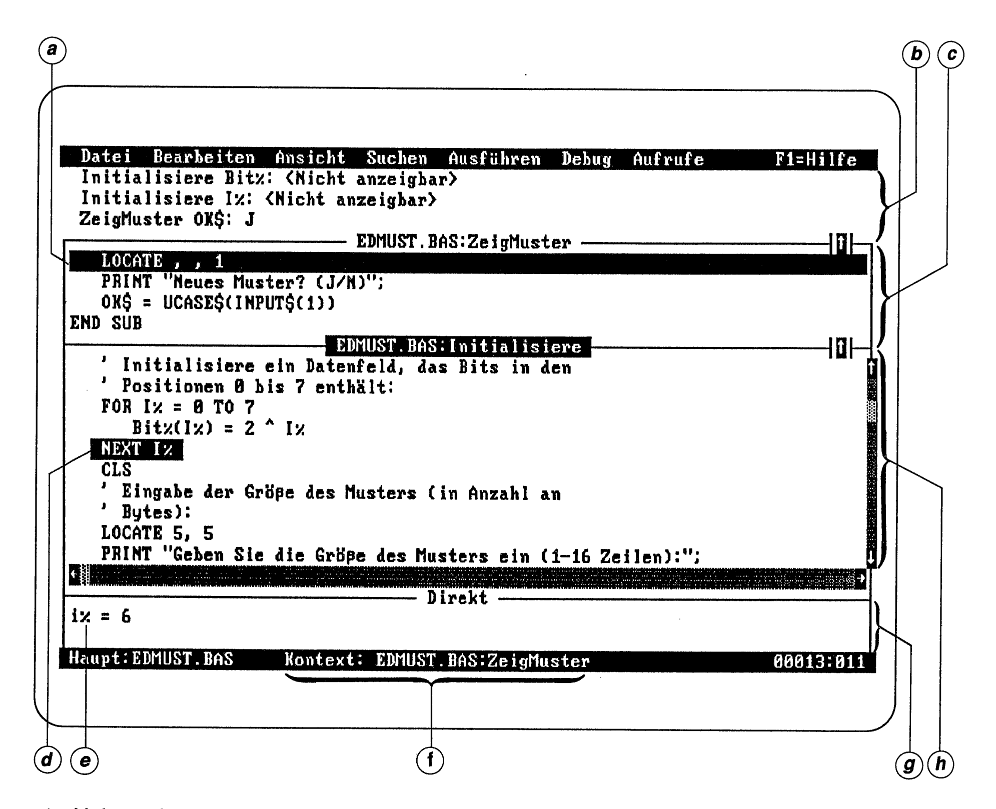
Sie können Haltepunkte und Stoppbedingungen setzen, um die Ausführung Ihres Programmes anzuhalten, wenn bestimmte Ereignisse oder Bedingungen auftreten. Sie können einen Anzeigeausdruck setzen, um die Werte von Variablen oder Ausdrücken zu verfolgen, währenddessen Ihr Programm ausgeführt wird.

7.3.4.1 Das Debug-Fenster

Wenn Sie eine Stoppbedingung setzen oder einen Anzeigeausdruck hinzufügen, öffnet sich das Debug-Fenster im oberen Teil des Bildschirms. Der Name jeder Variablen oder jeden Ausdrucks, die/den Sie verfolgen, erscheint in dem Debug-Fenster.

Abbildung 7.2 zeigt den QuickBASIC-Bildschirm mit Einträgen in dem Debug-Fenster, mit zwei Arbeitsbereichen sowie dem Direkt-Fenster.

Abbildung 7.2 Debug-Eigenschaften



a) Haltepunkt

b) Debug-Fenster

c) Der obere Arbeitsbereich enthält aktuell Modul-Ebenen-Code.

d) Nächste auszuführende Anweisung

e) Dieser Wert kann in dem aktuellen Kontext verändert werden.

f) Zeigt, welcher Programmteil gerade ausgeführt wird.

g) Direkt-Fenster

h) Der untere Arbeitsbereich enthält den aktuellen Prozedur-Ebenen-Code.

Das Debug-Fenster öffnet sich immer dann, wenn Sie entweder den Befehl **Variable anzeigen** oder den Befehl **Stoppbedingung** aus dem Menü **Debug** wählen. Jeder Eintrag in der Liste enthält die folgenden Informationen:

- Den Kontext des Ausdruckes.

  Der Kontext ist entweder ein Modulname (wenn Sie **Variable anzeigen** oder **Stoppbedingung** wählen, während sich Modul-Ebenen-Code in dem aktiven Arbeitsbereich befindet) oder ein Prozedurname (wenn Sie **Variable anzeigen** oder **Stoppbedingung** wählen, während sich Prozedur-Ebenen-Code in dem aktiven Arbeitsbereich befindet). Beachten Sie, daß der Kontext eines Ausdruckes, wie in dem Debug-Fenster gezeigt, gleich bleibt, obwohl der Kontext (der Ausführung), gezeigt auf der Statusleiste, entsprechend des Programmablaufes wechselt. Wenn diese beiden Zusammenhänge unterschiedlich sind, erscheint in dem Debug-Fenster die Meldung `Nicht anzeigbar`. Weitere Informationen zu dem Kontext eines Anzeigeausdrucks oder einer Stoppbedingung finden Sie in Abschnitt 7.3.1, "Debug-Begriffe und -Konzepte".
- Den Ausdruck, den Sie verfolgen möchten.
- Den Wert oder die Bedingung (WAHR oder FALSCH) des Ausdrucks, solange der Ablauf der Programmlogik sich innerhalb des Ausdruckzusammenhanges befindet. Beachten Sie, daß in dem Debug-Fenster der Variablen `Bit%` "`Nicht anzeigbar`" folgt und nicht ein Wert oder die Wörter WAHR oder FALSCH, da das in Abbildung 7.2 gezeigte Programm aktuell innerhalb der Prozedur `Initialisiere` ausgeführt wird. Dies ergibt sich, weil die Variable `Bit%` in das Debug-Fenster eingegeben wurde, während sich der Modul-Ebenen-Code von `EDMUST.BAS` in dem aktiven Arbeitsbereich befand. Daher hat der Debugger keinen Zugriff auf die Variable, während sich der aktuelle Kontext der Ausführung (gezeigt auf der Statusleiste) auf der Prozedur-Ebene befindet. Sowohl die Variable `OK$` als auch `i%` wurden jedoch in das Debug-Fenster eingegeben, während die Prozedur `Initialisiere` sich in dem aktiven Arbeitsbereich befand, so daß die aktuellen Werte dieser Variablen in dem Debug-Fenster erschienen.

Die Geschwindigkeit der Programmausführung verringert sich merklich, wenn sich in dem Debug-Fenster viele Positionen befinden.

Der obere Arbeitsbereich zeigt den aktuellen Modul-Ebenen-Code des Hauptmodules von `EDMUST.BAS`. Die in diesem Fenster hervorgehobene Zeile stellt einen Haltepunkt dar, der die Programmausführung anhält, sobald die Kontrolle aus der Prozedur Initialisiere zurückgegeben wird.

Der untere Arbeitsbereich zeigt den Code der Prozedur Initialisiere. Dieser zweite Arbeitsbereich ist nun das aktive Fenster. Die hervorgehobene Anweisung in diesem Fenster ist diejenige, die das Programm als nächste ausführen wird, und bestimmt daher den auf der Statusleiste gezeigten Kontext.

Anweisungen in dem Direkt-Fenster haben als Kontext die nächste in dem Programm auszuführende Anweisung. Falls der Benutzer sich an diesem Punkt in das Direkt-Fenster bewegte und die dort gezeigte Anweisung ausführte, würde der Schleifenzähler in der **FOR...NEXT**-Schleife der Prozedur `Initialisiere` den Wert 55 annehmen. Eine solche Technik ist praktisch, wenn das Bearbeiten des Programms im Arbeitsbereich Sie dazu zwingen würde, das Programm neu zu starten.

Die Statusleiste am Fuß des Bildschirms zeigt das aktuelle Stadium der Programmausführung an.

7.3.4.2 Haltepunkte (F9)

Ein Haltepunkt ist eine bestimmte Position in der Ablaufsteuerung Ihres Programmes, an der Sie die Ausführung anhalten möchten. Sie können Haltepunkte dazu einsetzen, Ihr Programm an zentralen Punkten zu unterbrechen, an denen Sie Probleme vermuten, und anschließend die Variablenwerte prüfen, um Ihre Vermutung zu bestätigen oder zu widerlegen.

Setzen oder löschen Sie einen Haltepunkt mit dem Befehl **Haltepunkt ein/aus** aus dem Menü **Debug** (oder betätigen Sie F9). Die Zeilen, in denen Haltepunkte gesetzt sind, werden in rot oder invertierter Darstellung gezeigt. Verwenden Sie den Befehl **Alle Haltepunkte löschen**, wenn Sie alle von Ihnen zuvor in einem Programm gesetzten Haltepunkte löschen möchten.

7.3.4.3 Stoppbedingungen

Setzen Sie eine Stoppbedingung mit dem Befehl **Stoppbedingung** aus dem Menü **Debug**. Verwechseln Sie den Befehl **Stoppbedingung** nicht mit dem Befehl **Variable anzeigen**, der den aktuellen Wert einer Variablen oder eines Ausdruckes während der Ausführung zeigt. (Eine Beschreibung des Befehls **Variable anzeigen** finden Sie in Abschnitt 7.3.4.4.)

Sie können jeden beliebigen Variablennamen oder Ausdruck in das Dialogfeld **Stoppbedingung** eingeben, und der Ausdruck darf jeden der QuickBASIC-Vergleichsoperatoren (wie zum Beispiel <>) enthalten. Wenn Sie lediglich sehen möchten, wann ein Ausdruck von Null auf ungleich Null wechselt, müssen Sie den Vergleichsoperator <> nicht ausdrücklich verwenden. Es genügt, den Ausdruck dem Debug-Fenster hinzuzufügen, und BASIC unterbricht die Ausführung des Programmes, sobald der Ausdruck einen Wert ungleich Null bekommt. (BASIC betrachtet immer einen Wert ungleich Null als wahr und einen Wert gleich Null als falsch. Weitere Informationen finden Sie in Kapitel 1, "Strukturen zur Ablaufsteuerung", in *Programmieren in BASIC: Ausgewählte Themen.*)

Falls Sie eine Stoppbedingung setzen, während sich eine **SUB**- oder **FUNCTION**-Prozedur in dem aktiven Arbeitsbereich befindet, beschreibt der Debugger den gegebenen Ausdruck als entweder WAHR oder FALSCH nur dann, wenn das Programm diese Prozedur ausführt. Wenn das Programm in einem anderen Kontext ausgeführt wird, erscheint neben dieser Position die Meldung `Nicht anzeigbar`.

Stoppbedingungen sind besonders hilfreich, wenn Sie zwar wissen, daß eine Variable einen unerwarteten Wert annimmt, Sie aber nicht wissen, wo dies in dem Ablauf Ihres Programmes geschieht. Für ein BASIC-Programm mit vielen **GOSUB**-Unterroutinen ist es sehr einfach, den Wert einer Variablen zu ändern. Wenn zum Beispiel die Unterroutine `AktualisiereFenster` in der folgenden Schleife unbeabsichtigt eine Variable mit dem Namen `I%` verwendet, dann wirkt sich jede Änderung, die in `AktualisiereFenster` auftritt, auch auf diese Schleife aus:

```
FOR I% = 1 TO 20
   GOSUB AktualisiereFenster
NEXT I%
```

Falls die Schleife unerwartet endet, kann der Benutzer in diesem Fall die Stoppbedingung `I% <> 1` hinzufügen und anschließend die Ausführung durch das Betätigen von F5 fortsetzen. Sobald der Ausdruck `I% <> 1` wahr wird, hält die Programmausführung an, und der Benutzer kennt genau die Zeile, in der die Variable unbeabsichtigt verändert wird.

7.3.4.4 Anzeigeausdrücke: Mehr als PRINT

Verwenden Sie den Befehl **Variable anzeigen** aus dem Menü **Debug**, um in dem Debug-Fenster die aktuellen Werte von Variablen oder Ausdrücken auszugeben.

Die Fähigkeit, Ausdrücke zu beobachten, bewahrt Sie vor der Notwendigkeit, bei der Verfolgung eines Variablenwertes wiederholt **PRINT**-Anweisungen einzusetzen.

Wenn Sie einen Ausdruck während der Programmausführung beobachten, sehen Sie umgehend die Änderungen des Wertes. Manchmal folgen dem Namen des Ausdrucks anstelle eines Wertes die Meldung `Nicht anzeigbar`. Dies bedeutet, daß der Debugger zu diesem Zeitpunkt keinen Zugriff auf die Variable hat, da die Variable in das Debug-Fenster eingegeben wurde, als sich ein anderer Teil des Programmes im aktiven Arbeitsbereich befanden.

Meistens ist es nicht notwendig, das Debug-Fenster zu benutzen. Wenn Sie nur in einem Fall an dem Wert einer Variablen interessiert sind, genügt es, deren Namen in dem Direkt-Fenster als Argument einer **PRINT**-Anweisung einzugeben.

Beispiel

Das folgende Beispiel berechnet die *n*-te Zahl in einer Folge, die als Fibonacci-Folge bezeichnet wird. In einer Fibonacci-Folge ist jede Zahl die Summe der beiden vorhergehenden Zahlen. Die ersten beiden Zahlen in der Folge sind 1 und 2. Es folgen die ersten zehn Zahlen der Reihe:

1, 2, 3, 5, 8, 13, 21, 34, 55, 89

Folgen Sie diesen Schritten, um eine Demonstration animierter Verfolgung mit einem Anzeigeausdruck zu sehen:

1. Starten Sie QuickBASIC, wählen Sie anschließend den Befehl **Neues Programm** aus dem Menü **Datei**, und geben Sie folgende Zeilen ein:

```
DEFINT A-Z
INPUT N
PRINT Fib(N)
END
```

2. Wählen Sie den Befehl **Variable anzeigen** aus dem Menü **Debug**, und geben Sie anschließend N in das Textfeld ein. Dies zeigt den Wert an, den N auf Modul-Ebene hat.
3. Erstellen Sie eine als "`Fib`" benannte **FUNCTION**-Prozedur, indem Sie den Befehl **Neue FUNCTION** aus dem Menü **Bearbeiten** wählen. (Beachten Sie, daß DEFINT A-Z automatisch wegen der Anweisung **DEF***Typ*, die Sie in Schritt 1 eingegeben haben, über der ersten Zeile der **FUNCTION**-Definition eingefügt wird.) Fügen Sie nun Zeilen ein, so daß die Prozedur wie folgt erscheint:

```
FUNCTION Fib (N)
   IF N > 2 THEN
      Fib = Fib (N - 1) + Fib(N - 2)
   ELSE
      Fib = N
   END IF
END FUNCTION
```

4. Wählen Sie den Befehl **Variable anzeigen** aus dem Menü **Debug**, fügen Sie anschließend N in das Debug-Fenster ein. Dies zeigt den Wert an, den N innerhalb der Prozedur `Fib` hat.

5. Schalten Sie den Befehl **Verfolgen ein** des Menüs **Debug** ein.
6. Wählen Sie **Start** aus dem Menü **Ausführen** (oder betätigen Sie UMSCHALTTASTE+F5), um das Programm auszuführen.
7. Geben Sie nach der DOS-Eingabeaufforderung den Wert 8 ein.
8. Beobachten Sie das Debug-Fenster, um zu sehen, wie sich der Wert von N ändert, wenn sich die Prozedur rekursiv aufruft, um die achte Zahl der Fibonacci-Folge zu berechnen. (Weitere Informationen zu rekursiven Prozeduren finden Sie in Kapitel 2, "Prozeduren: Unterprogramme und Funktionen", in *Programmieren in BASIC: Ausgewählte Themen.*)
9. Wählen Sie den Befehl **Ausgabebildschirm** aus dem Menü **Ansicht** (oder betätigen Sie F4), um den berechneten Endwert zu sehen.

Um zu sehen, wie die Tasten Rückverfolgung nach oben und Rückverfolgung nach unten arbeiten, folgen Sie mit dem vorhergehenden Programm diesen Schritten:

1. Schalten Sie den Befehl **Verfolgen ein** des Menüs **Debug** aus.
2. Schalten Sie den Befehl **Rückverfolgen ein** des Menüs **Debug** ein.
3. Setzen Sie einen Haltepunkt an der **END FUNCTION**-Anweisung.
4. Starten Sie das Programm.
5. Betätigen Sie UMSCHALTTASTE+F8, um die letzten 20 Anweisungen nach oben durchzusehen; wenn die Ausführung an dem von Ihnen gesetzten Haltepunkt anhält, betätigen Sie UMSCHALTTASTE+F10, um dieselben 20 Anweisungen nach unten durchzusehen.

7.3.5 Wie Sie die Ausführung steuern

Mit den QuickBASIC-Befehlen zur Ausführungssteuerung können Sie Ihr Programm starten, dessen Ausführung anhalten und anschließend fortfahren. Diese Befehle ermöglichen es Ihnen, eine Anweisung nach der anderen auszuführen, entweder mit oder ohne Ausführung der dazwischenliegenden Anweisungen.

7.3.5.1 Weiter (F5)

Wählen Sie den Befehl **Weiter** aus dem Menü **Ausführen**, oder betätigen Sie F5, um Ihr Programm zu starten. Wenn Ihr Programm startet, wird die Ausführung entweder mit der Anweisung fortgesetzt, die der letzten ausgeführten Anweisung folgt, falls die Ausführung aktuell unterbrochen ist, oder startet vom Beginn. Der Status eines Programmes wird in der Mitte der Statusleiste unten auf dem QuickBASIC-Bildschirm hinter dem Wort Kontext gezeigt. Falls das Programm läuft oder unterbrochen ist, zeigt Kontext das aktuelle Modul (sowie die Prozedur, sofern vorhanden) an. Falls das Programm nicht läuft, wird dies angezeigt mit "Kontext: Programm wird nicht ausgeführt".

Falls der Befehl **Verfolgen ein** eingeschaltet ist, wird das Programm in Zeitlupen-Animation ausgeführt; andernfalls wird es mit normaler Geschwindigkeit ausgeführt. Falls das Debug-Fenster geöffnet oder der Schalter **Rückverfolgen ein** eingeschaltet ist, wird das Programm zwar mit geringerer als normaler Geschwindigkeit ausgeführt, die Programmverfolgung wird aber solange nicht angezeigt, bis auch **Verfolgen ein** eingeschaltet ist.

Sie benutzen den Befehl **Weiter** im allgemeinen, um ein Programm zu starten und anschließend von einem Haltepunkt oder einer Stoppbedingung aus bis zu einem/einer anderen auszuführen. Solange die Ausführung angehalten ist, können Sie den Code untersuchen und, falls notwendig, Berichtigungen vornehmen. Wenn die Korrekturen durchgeführt sind, können Sie F5 abermals betätigen, um die Ausführung an diesem Punkt fortzusetzen. Sollte die Änderung Ihres Programmtextes derart sein, daß das Programm nicht mit der Ausführung fortfahren kann, wenn es die von Ihnen durchgeführten Änderungen einbindet, gibt QuickBASIC eine Warnung aus und fragt Sie, ob Sie ohne die Änderungen fortfahren möchten oder das Programm mit den Änderungen vom Beginn neu starten möchten. Falls Sie nicht neu starten möchten, können Sie notwendige Änderungen an Programmdaten normalerweise temporär in dem Direkt-Fenster vornehmen und anschließend fortfahren.

7.3.5.2 Ausführen bis zum Cursor (F7)

Das Betätigen der Taste F7 führt Ihr Programm von der aktuellen Zeile bis zu der Zeile aus, auf der der Cursor steht, vorausgesetzt, daß sich die Zeile, auf der der Cursor steht, im Ablauf der Programmausführung befindet. Das heißt, falls der Cursor sich auf einer unerreichbaren Zeile (zum Beispiel innerhalb einer Prozedur, die von dem Programm niemals aufgerufen wird, oder einer Zeile, die von dem Programm bereits ausgeführt wurde) befindet, dann ist die Betätigung von F7 gleichbedeutetend mit der Betätigung von F5 (Fortfahren mit der Anweisung hinter der letzten ausgeführten Anweisung).

Hinweis Das Betätigen des rechten Mausknopfes führt Ihr Programm bis zu der Zeile aus, auf der der Mauszeiger steht.

7.3.5.3 Nächste Anweisung festlegen

Der Befehl **Nächste Anweisung festlegen** aus dem Menü **Debug** ändert die Reihenfolge der Ausführung Ihres Programmes so, daß die nächste ausgeführte Anweisung diejenige ist, auf der sich der Cursor befindet. Der Befehl führt das Programm nicht bis zu dieser Anweisung aus, sondern springt einfach dorthin, ohne den dazwischenliegenden Code auszuführen. Dieser Effekt ist ähnlich dem einer **GOTO**-Anweisung.

Dieselben Einschränkungen, die für die Anweisung **GOTO** gelten, gelten auch für den Befehl **Nächste Anweisung festlegen**. Zum Beispiel können Sie nicht aus der Modul-Ebene des Programmes heraus in die Mitte einer Prozedur verzweigen.

7.3.5.4 Nächste Anweisung

Der Befehl **Nächste Anweisung** aus dem Menü **Ansicht** plaziert den Cursor auf der nächsten auszuführenden Anweisung, sobald das Programm mit der Ausführung fortfährt. Der Befehl führt die Anweisung nicht aus. **Nächste Anweisung** ist hilfreich, wenn Sie ein unterbrochenes Programm untersucht haben und zu dem Punkt der Unterbrechung zurückkehren möchten, ohne die Anweisung auszuführen. In einem Programm, das gerade nicht läuft, plaziert der Befehl **Nächste Anweisung** den Cursor einfach auf der ersten Programmanweisung.

7.3.5.5 Neustart

Der Befehl **Neustart** aus dem Menü **Ausführen** initialisiert alle Variablen auf Null oder Nullzeichenketten und hebt in Ihrem Programm die erste ausführbare Anweisung hervor. Sie können ihn grundsätzlich benutzen, wenn Sie mit der Ausführung des Programmes nicht fortfahren wollen. Dieser Befehl ist besonders hilfreich, wenn Sie ein Programm von Beginn an neu starten möchten und den Code durchlaufen möchten, der vor dem ersten Haltepunkt oder der ersten Stoppbedingung steht. Wenn Sie nur neu starten möchten und bis zu dem ersten Haltepunkt oder zur ersten Stoppbedingung ausführen lassen möchten, wählen Sie den Befehl **Start** aus dem Menü **Ausführen**.

7.4 Fortgeschrittenes Debuggen

Während die Komplexität Ihrer Programme zunimmt, werden Sie weitere Möglichkeiten kennenlernen, die Debug-Eigenschaften von QuickBASIC zu verwenden. Zwei der leistungsfähigsten sind das Menü **Aufrufe** und die Fähigkeit, ausführbare Dateien zu erstellen, die kompatibel mit dem Microsoft CodeView-Debugger sind.

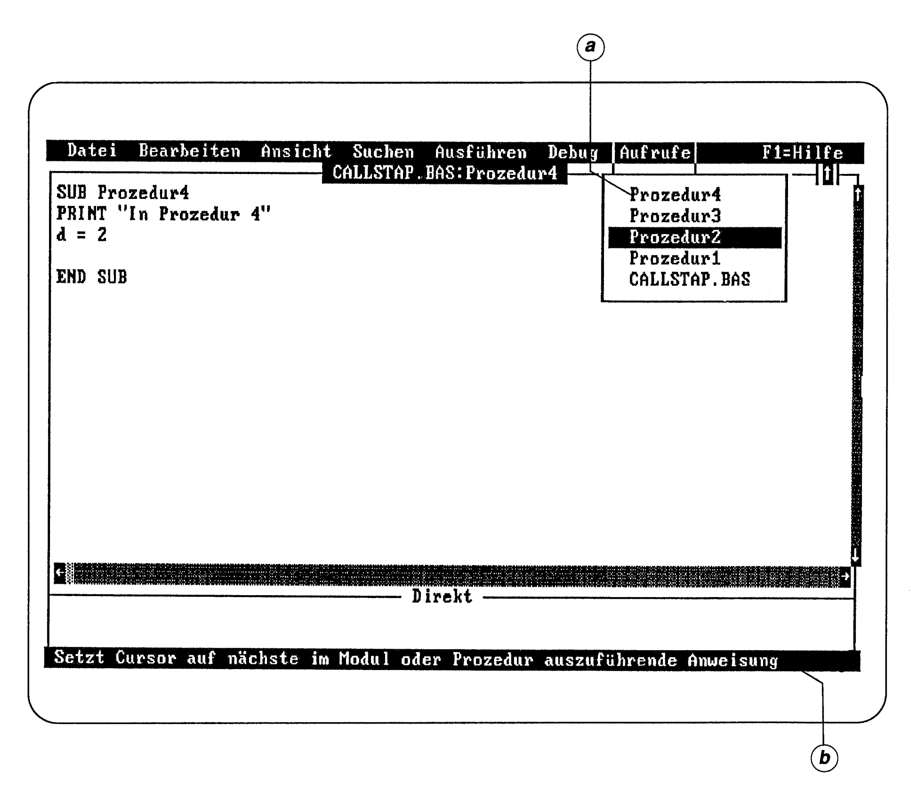
7.4.1 Das Menü Aufrufe

Das Menü **Aufrufe** unterscheidet sich von anderen QuickBASIC-Menüs darin, daß es keine Befehle enthält. Falls Ihr Programm keine Prozeduraufrufe aus anderen Prozeduren heraus enthält, zeigt dieses Menü außer dem Hauptmodulnamen Ihres Programmes und den Namen der Prozedur, die aktuell ausgeführt wird, niemals etwas an.

Falls Ihr Programm Prozeduren aus anderen Prozeduren heraus aufruft, zeigt das Menü **Aufrufe** solche Aufrufe zwischen Prozeduren an. Zum Beispiel könnten Sie eine Abfolge von Aufrufen haben, in denen der Modul-Ebenen-Code (hier dargestellt von dem Programmnamen CALLSTAP) die `Prozedur1` aufruft, `Prozedur1` anschließend `Prozedur2` aufruft, `Prozedur2` `Prozedur3` aufruft und `Prozedur3` `Prozedur4` aufruft. Unter diesen Umständen würde das Menü **Aufrufe** so aussehen wie dasjenige in Abbildung 7.3.



*Abbildung 7.3 Abfolge von Prozeduren im Menü **Aufrufe***



a) Prozedur, die aktuell ausgeführt wird.

b) Beschreibt den Effekt der Wahl der hervorgehobenen Prozedur aus dem Menü.

Die Prozedur oben auf dem Menü ist die aktuell aktive Prozedur, und diejenige direkt darunter ist die Prozedur, zu der die erste Prozedur zurückkehrt, wenn sie Ihre **END SUB**- oder **END FUNCTION**-Anweisung ausführt. Sie können Ihr Programm bis zu jeder dieser Prozeduren ausführen, indem Sie einfach den Prozedurnamen aus dem Menü **Aufrufe** wählen und anschließend F7 betätigen. Da `Prozedur2` in der Abbildung hervorgehoben ist, würde die Betätigung der EINGABETASTE und anschließend die Betätigung von F7 allen Code zwischen dem Eintrag von `Prozedur4` oben auf der Liste und der nächsten in `Prozedur2` auszuführenden Anweisung ausführen.

Manchmal wird die Liste in dem Menü **Aufrufe** als Stapel bezeichnet, da die oberste Prozedur die zuletzt aufgerufene ist. Das Menü **Aufrufe** kann acht Prozeduren anzeigen, die Anzahl an Aufrufen in Ihrem Programm zwischen Prozeduren ist aber nur durch die verfügbare Größe des Stapelplatzes begrenzt. Wenn mehr als acht solcher Aufrufe vorhanden sind, zeigt das Menü **Aufrufe** nur die letzten acht. Um einen der davorliegenden Aufrufe weiter unten im Stapel zu sehen, müssen Sie rückwärts ausführen lassen.

Sie können mit den Eigenschaften des Aufrufstapels experimentieren, indem Sie das Fibonacci-Beispiel am Ende von Abschnitt 7.3.4.4, "Anzeigeausdrücke: Mehr als PRINT", verwenden. Da die Prozedur `Fib` rekursiv ist, legt die Zahl, die Sie bei der Anfrage eingeben, die Höhe des Aufrufstapels und die Anzahl der Prozeduren in dem Menü **Aufrufe** fest. Nach der Unterbrechung der Ausführung (entweder mit einer Stoppbedingung oder durch die Betätigung von STRG+UNTBR) können Sie das Menü **Aufrufe** anzeigen und die oben beschriebenen Methoden verwenden, um zwischen verschiedenen Aufrufen der Prozedur `Fib` auszuführen.

7.4.2 Kompatibilität mit dem CodeView-Debugger

Sie können Routinen aus anderen Sprachen in Ihre QuickBASIC-Programme einbinden, indem Sie diese in Quick-Bibliotheken einfügen. Obwohl Sie nicht die Debug-Eigenschaften von QuickBASIC zur Verfolgung durch Prozeduren, die sich in einer Quick-Bibliothek befinden, einsetzen können, können Sie den Microsoft CodeView-Debugger verwenden, um ausführbare Dateien zu debuggen, die Sie mit dem Befehlszeilen-Compiler BC erstellt haben. Um CodeView-kompatible ausführbare Dateien zu erstellen, müssen Sie für den **bc**-Befehl die Option **/zi** und für den **link**-Befehl die Option **/CO** verwenden.

Der CodeView-Debugger ist ein sehr leistungsfähiges Hilfsprogramm. Er wird mit dem Microsoft Macro Assembler (Version 5.0) und Microsoft C (Version 5.0) geliefert. Mit dem CodeView-Debugger können Sie Programme debuggen, die Routinen aus Microsoft BASIC, FORTRAN, Pascal, C und Macro Assembler kombinieren. Der Debugger zeigt den Quellcode jedes Moduls - sowohl in seiner ursprünglichen Quellform als auch in seiner Assembler-Form. Weitere Informationen zu Quick-Bibliotheken finden Sie in Kapitel 8, "Quick-Bibliotheken", weitere Informationen zu Objektdateien sowie zum Kompilieren und Binden von der Befehlszeile aus, um eine CodeView-Kompatibilität zu erhalten, finden Sie in Kapitel 9, "Wie Sie aus DOS heraus kompilieren und binden".

Weitere Informationen, wie Ihre BASIC-Programme in C oder Assembler geschriebene Routinen aufrufen können, finden Sie in Anhang C, "Wie Sie C- oder Assembler-Routinen aufrufen".

Hinweis Aus der QuickBASIC-Umgebung heraus können Sie keine CodeView-kompatible ausführbare Datei erzeugen. Die Option Debug-Code erstellen aus dem Dialogfeld **EXE-Datei erstellen** läßt eine ausführbare Datei nicht CodeView-kompatibel werden.

Teil 3: Bibliotheken und Hilfsprogramme

Dieser Abschnitt des Handbuches erläutert erweiterte QuickBASIC-Eigenschaften, wie zum Beispiel Bibliotheken und separate Kompilierung. Eine Bibliothek ist eine spezielle Datei, die alle Prozeduren eines Moduls oder einer Gruppe von Modulen enthält. Sie werden lernen, wie Sie sowohl Quick-Bibliotheken als auch selbständige Bibliotheken erstellen und benutzen, und wie Routinen, die in anderen Microsoft-Sprachen geschrieben sind, in Bibliotheken eingebunden werden. Außerdem werden Sie lernen, wie Sie von der DOS-Befehlszeile aus kompilieren, binden und ausführbare Dateien erstellen, die zu dem Microsoft CodeView-Debugger kompatibel sind.



8 Quick-Bibliotheken

Dieses Kapitel beschreibt, wie Sie Bibliotheken aus der QuickBASIC-Programmierumgebung heraus anlegen und pflegen. Eine Bibliothek ist eine Datei, die die Inhalte mehrerer Module unter einem einzigen Dateinamen enthält. Wenn Sie dieses Kapitel gelesen haben, werden Sie wissen, wie

- Bibliotheken aus der QuickBASIC-Umgebung heraus erstellt werden.
- Eine Quick-Bibliothek geladen wird, wenn ein QuickBASIC-Programm läuft.
- Sie sich den Inhalt einer Quick-Bibliothek ansehen können.
- Routinen, die in anderen Sprachen geschrieben sind, in eine Quick-Bibliothek eingefügt werden.

8.1 Bibliotheksarten

QuickBASIC bietet Hilfsprogramme zur Erstellung zweier verschiedener Bibliotheksarten, die mit unterschiedlichen Erweiterungen der Dateinamen gekennzeichnet werden:

Erweiterung	*Funktion*
.QLB	Die Erweiterung .QLB kennzeichnet eine Quick-Bibliothek, eine besondere Art einer Bibliothek, die es ermöglicht, leicht Erweiterungen der Sprache BASIC vorzunehmen, die innerhalb der QuickBASIC-Programmierumgebung erkannt werden. Eine Quick-Bibliothek kann Prozeduren enthalten, die in QuickBASIC oder anderen Microsoft-Sprachen geschrieben sind, wie zum Beispiel Microsoft C.
.LIB	Die Erweiterung .LIB kennzeichnet eine Bibliothek, die als selbständige (.LIB) Bibliothek bezeichnet wird und mit dem Microsoft Library Manager, LIB, erstellt ist. Wenn QuickBASIC eine Quick-Bibliothek anlegt, legt es gleichzeitig eine .LIB-Bibliothek an, die dieselben Prozeduren in leicht abgewandelter Form enthält.

Sie können beide Bibliotheksarten aus der Programmierumgebung heraus oder von der Befehlszeile aus anlegen. Sie können sich eine Quick-Bibliothek dann als eine Gruppe von Prozeduren vorstellen, die in QuickBASIC "eingebunden" werden, wenn die Bibliothek mit QuickBASIC geladen wird. Bibliotheken mit der Erweiterung .LIB sind im wesentlichen kompilierte aber nicht gebundene Prozeduren. Sie können entweder einer Quick-Bibliothek hinzugefügt werden oder mit einem Hauptmodul gebunden werden, um eine Datei zu erzeugen, die von der DOS-Befehlszeile aus gestartet werden kann.

Dieses Kapitel erläutert die Verwendung von Befehlszeilen-Hilfsprogrammen für einige allgemeine Fälle. Für eine umfassende Erläuterung zur Verwendung solcher Hilfsprogramme sollten Sie jedoch in Kapitel 9, "Wie Sie aus DOS heraus kompilieren und binden", nachschlagen.

8.2 Vorteile von Quick-Bibliotheken

Quick-Bibliotheken erleichtern in vieler Hinsicht die Programmentwicklung bzw. -pflege.

Währenddessen die Entwicklung eines Projektes fortschreitet, und Module fester Bestandteil Ihres Programmes werden, können Sie diese in eine Quick-Bibliothek einfügen und anschließend die Quelldateien für die Originalmodule beiseite legen, bis Sie diese Quelldateien verbessern oder pflegen möchten. Danach können Sie die Bibliothek mit QuickBASIC laden, und Ihr Programm hat sofortigen Zugriff auf alle Prozeduren in der Bibliothek.

Prozeduren in einer Quick-Bibliothek verhalten sich wie QuickBASIC-Anweisungen. Wenn sie richtig deklariert ist, kann eine **SUB**-Prozedur in der Quick-Bibliothek sogar ohne eine **CALL**-Anweisung aufgerufen werden, so daß die Prozedur wie eine BASIC-Anweisung aussieht. Weitere Informationen zum Aufruf einer **SUB**-Prozedur mit oder ohne Schlüsselwort **CALL** finden Sie im Kapitel 2, "Prozeduren: Unterprogramme und Funktionen", in *Programmieren in BASIC: Ausgewählte Themen.*

Prozeduren einer Quick-Bibliothek können direkt aus dem Direkt-Fenster heraus ausgeführt werden, genauso wie BASIC-Anweisungen. Das bedeutet, daß Sie deren Auswirkungen testen können, bevor Sie sie in anderen Programmen einsetzen.

Sie können Routinen, die in anderen Sprachen geschrieben sind, in Quick-Bibliotheken einbinden. Zum Beispiel könnten Sie eine Bibliothek anlegen, die C-Funktionen und Assembler-Routinen sowie BASIC-Prozeduren enthält. Alle diese Routinen verhalten sich wie tatsächliche Erweiterungen der Sprache QuickBASIC, wenn Sie diese Bibliothek mit QuickBASIC laden.

Wenn Sie zusammen mit anderen Programmierern Programme entwickeln, erleichtern es Quick-Bibliotheken, eine Zusammenstellung gemeinsamer Prozeduren zu aktualisieren. Ähnlich werden, wenn Sie eine Bibliothek mit Originalprozeduren zur kommerziellen Nutzung anbieten möchten, alle QuickBASIC-Programmierer in der Lage sein, diese Prozeduren sofort für eigene Anwendungen einzusetzen. Sie könnten Ihre selbsterstellte Quick-Bibliothek möglichen Kunden vor dem Kauf für Demozwecke zur Verfügung stellen. Da Quick-Bibliotheken keinen Quellcode enthalten und nur innerhalb der QuickBASIC-Programmierumgebung verwendet werden können, werden Ihre Urheberrechte geschützt, während Ihre Marktchancen verbessert werden.

Hinweis Quick-Bibliotheken haben dieselbe Funktion wie Benutzer-Bibliotheken in QuickBASIC Version 2.0 bzw. 3.0. Eine Benutzer-Bibliothek können Sie jedoch nicht als Quick-Bibliothek laden. Sie müssen die Bibliothek mit dem ursprünglichen Quellcode neu anlegen, wie unten beschrieben.

8.3 Wie Sie eine Quick-Bibliothek anlegen

Eine Quick-Bibliothek enthält automatisch alle Module, sowohl das Hauptmodul als auch die weiteren Module, die sich innerhalb der QuickBASIC-Umgebung befinden, wenn Sie die neue Bibliothek anlegen. Die Bibliothek enthält darüber hinaus den Inhalt jeder anderen Quick-Bibliothek, die Sie beim Start von QuickBASIC geladen haben. Falls Sie ein ganzes Programm laden, aber nur bestimmte Module in der Bibliothek ablegen möchten, müssen Sie diejenigen, die Sie nicht verwenden möchten, ausdrücklich entfernen. Module können Sie mit dem Befehl **Datei entfernen** aus dem Menü **Datei** entfernen.

Sie können schnell überblicken, welche Module geladen sind, indem Sie sich das Listenfeld des Befehls **SUBs** aus dem Menü **Ansicht** ansehen. Diese Methode zeigt Ihnen jedoch nicht, welche Prozeduren eine geladene Bibliothek enthält. Das Hilfsprogramm QLBDUMP.BAS, beschrieben in Abschnitt 8.4.3, "Wie Sie sich den Inhalt einer Quick-Bibliothek ansehen", erlaubt es Ihnen, alle Prozeduren einer Bibliothek aufzulisten.

Nur ganze Module können in eine Bibliothek geschrieben werden. Das bedeutet, Sie können nicht einige Prozeduren eines Moduls auswählen. Wenn Sie nur bestimmte Prozeduren aus einem Modul einbinden möchten, schreiben Sie die von Ihnen gewünschten Prozeduren in ein separates Modul und legen dieses Modul anschließend in einer Bibliothek ab.

Eine Quick-Bibliothek muß abgeschlossen sein. Das heißt, sie darf keinen Bezug auf eine Prozedur außerhalb der Bibliothek nehmen. Daher müssen alle von einer Prozedur der Quick-Bibliothek aufgerufenen Prozeduren auch in der Quick-Bibliothek abgelegt werden.

Bei großen Programmen können Sie die Ladezeit reduzieren, indem Sie so viele Routinen wie möglich in Quick-Bibliotheken ablegen. Das Ablegen vieler Routinen in Quick-Bibliotheken ist darüber hinaus ein Vorteil, wenn Sie vorhaben, das Programm später in eine selbständig ausführbare Datei zu überführen, da der Inhalt von Bibliotheken einfach eingebunden wird, ohne erneut kompiliert zu werden.

Hinweis Ihr Hauptmodul kann Prozeduren enthalten. Wenn dies der Fall ist, und Sie diese Prozeduren in die Bibliothek einbinden, wird auch das gesamte Hauptmodul in der Bibliothek abgelegt. Dies erzeugt zwar keine Fehlermeldung, aber der Modul-Ebenen-Code in der Bibliothek kann niemals ausgeführt werden, es sei denn, eine ihrer Prozeduren enthält eine Routine (wie zum Beispiel **ON ERROR**), die die Kontrolle ausdrücklich an die Modul-Ebene übergibt. Selbst wenn dies der Fall ist, könnte vieles im Modul-Ebenen-Code unwesentlich sein. Wenn Sie Ihre Prozeduren in Modulen aufbauen, die häufig zusammen benutzt werden, werden Ihre Quick-Bibliotheken wahrscheinlich weniger überflüssigen Code enthalten.

8.3.1 Dateien, die zum Anlegen einer Quick-Bibliothek benötigt werden

Um eine Quick-Bibliothek anzulegen, müssen Sie vor dem Beginn sicherstellen, daß die richtigen Dateien zur Verfügung stehen. Wenn Sie keine Festplatte einsetzen, sollten Sie Ihre Dateien sowie die weiteren Programme auf verschiedenen Disketten ablegen. QuickBASIC verlangt von Ihnen die Angabe eines Pfadnamens, wenn es eine Datei nicht finden kann: Sollte dies passieren, legen Sie die richtige Diskette ein, und antworten Sie auf die Anfrage.

Stellen Sie sicher, daß sich die folgenden Dateien im aktuellen Arbeitsverzeichnis befinden oder für QuickBASIC durch die entsprechenden DOS-Umgebungsvariablen verfügbar sind:

Datei	***Zweck***
QB.EXE	Steuert den Prozeß der Erstellung einer Quick-Bibliothek. Falls Sie nur mit QuickBASIC-Modulen arbeiten, können Sie alles in einem Schritt innerhalb der QuickBASIC-Umgebung ausführen.
BC.EXE	Erstellt Objektdateien aus dem Quellcode.
LINK.EXE	Bindet Objektdateien.
LIB.EXE	Verwaltet selbständige Bibliotheken von Objektmodulen.
BQLB40.LIB	Stellt Routinen zur Verfügung, die Ihre Quick-Bibliothek benötigt. Diese Bibliothek ist eine selbständige Bibliothek, die mit Objekten aus Ihrer Bibliothek gebunden wird, um eine Quick-Bibliothek zu bilden.

8.3.2 Wie Sie eine Quick-Bibliothek aufbauen

Meistens legen Sie Quick-Bibliotheken innerhalb der QuickBASIC-Umgebung an. Gelegentlich möchten Sie vielleicht eine Bibliothek aktualisieren oder Routinen aus anderen Microsoft-Sprachen in die Bibliothek einbinden. In diesem Fall beginnen Sie mit der Erstellung einer Basis-Bibliothek aus den Nicht-BASIC-Routinen außerhalb der Umgebung, indem Sie LINK und LIB direkt aufrufen. Anschließend können Sie die aktuellsten QuickBASIC-Module innerhalb von QuickBASIC hinzufügen.

8.3.3 Wie Sie eine Quick-Bibliothek innerhalb der Umgebung aufbauen (Bibliothek erstellen)

Wenn Sie eine Bibliothek innerhalb der QuickBASIC-Umgebung erstellen, ist die erste Überlegung, ob es sich um eine neue Bibliothek oder ob es sich um eine Aktualisierung einer bereits existierenden Bibliothek handelt. Wenn es eine Aktualisierung ist, sollten Sie QuickBASIC mit der Befehlszeilen-Option **/l** starten und den Namen der zu aktualisierenden Bibliothek als Befehlszeilen-Argument angeben. Gleichzeitig können Sie auch den Namen eines Programmes angeben, dessen Module Sie in der Bibliothek ablegen möchten. In diesem Fall lädt QuickBASIC alle Module, die in der zu diesem Programm gehörenden .MAK-Datei angegeben sind.

Falls Sie Ihr Programm beim Start von QuickBASIC laden, stellen Sie sicher, daß Sie alle Module einschließlich des Hauptmoduls (es sei denn, es enthält Prozeduren, die Sie in der Bibliothek ablegen möchten) entfernen, die Sie nicht in der Quick-Bibliothek haben möchten.

Folgen Sie diesen Schritten, um Module zu entfernen:

1. Wählen Sie den Befehl **Datei entfernen** aus dem Menü **Datei.**
2. Markieren Sie im Listenfeld das Modul, das Sie entfernen möchten, betätigen Sie anschließend die EINGABETASTE.
3. Wiederholen Sie die Schritte 1 bis 2, bis Sie alle unerwünschten Module entfernt haben.

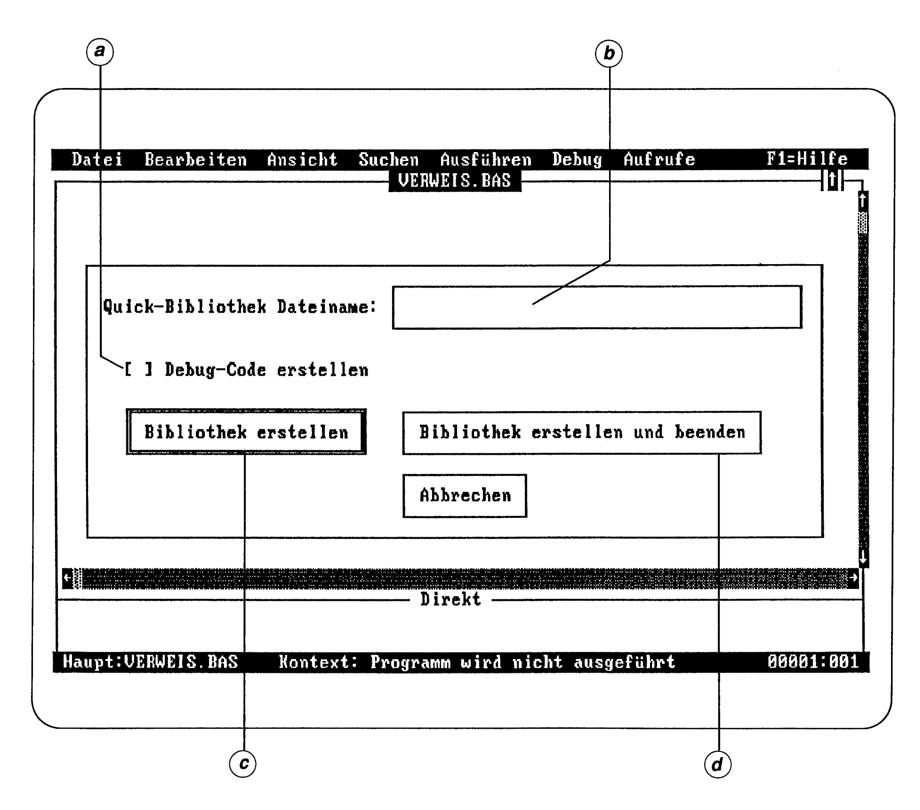
Alternativ dazu können Sie QuickBASIC einfach mit oder ohne eine Bibliotheksangabe starten und die von Ihnen gewünschten Module aus der Umgebung heraus nacheinander laden. In diesem Fall laden Sie jedes Modul mit dem Befehl **Datei laden** aus dem Menü **Datei.**

Um Module nacheinander mit QuickBASIC zu laden:

1. Wählen Sie den Befehl **Datei laden** aus dem Menü **Datei.**
2. Markieren Sie im Listenfeld den Namen eines Moduls, das Sie laden möchten.
3. Wiederholen Sie die Schritte 1 bis 2, bis Sie alle gewünschten Module geladen haben.

Nachdem Sie die vorhergehende Bibliothek (sofern vorhanden) sowie alle neuen Module, die Sie in die Quick-Bibliothek einbinden möchten, einmal geladen haben, wählen Sie den Befehl **Bibliothek erstellen** aus dem Menü **Ausführen**. Es erscheint das in Abbildung 8.1 gezeigte Dialogfeld.

Abbildung 8.1 Das Dialogfeld Bibliothek erstellen



a) Markieren Sie dieses Kontrollfeld, um Debug-Code einzubinden.

b) Geben Sie den Namen der zu erstellenden Bibliothek ein.

c) Wählen Sie diese Option, um eine Bibliothek aufzubauen und in QuickBASIC zu bleiben.

d) Wählen Sie diese Option, um eine Bibliothek aufzubauen und zu DOS zurückzukehren.

Um eine Quick-Bibliothek zu erstellen, müssen Sie die folgenden Schritte durchführen:

1. Geben Sie den Namen der Bibliothek, die Sie anlegen möchten, in das Textfeld "Quick-Bibliothek Dateiname" ein.

   Falls Sie nur einen Basisnamen angeben (das heißt, einen Dateinamen ohne Erweiterung), fügt QuickBASIC automatisch die Erweiterung .QLB an, wenn es die Bibliothek anlegt. Wenn Sie wünschen, daß Ihre Bibliothek keine Erweiterung hat, müssen Sie an den Basisnamen einen abschließenden Punkt (.) anfügen. Andernfalls können Sie jeden beliebigen Basisnamen und jede beliebige Erweiterung eingeben, der bzw. die den Regeln für DOS-Dateinamen entspricht.
2. Markieren Sie das Kontrollfeld "Debug-Code erstellen" nur, wenn Sie versuchen, einen Fehler aufzuspüren, den Sie in einer Bibliothek vermuten, die Sie gerade aktualisieren. Dies vergrößert Ihre Bibliothek, macht Ihr Programm langsamer und führt nur zu einem geringen Maß an Fehlerkontrolle, die sich hauptsächlich mit der Überprüfung von Datenfeldgrenzen beschäftigt.
3. Legen Sie die Quick-Bibliothek an:
   - Wählen Sie die Befehlsfläche "Bibliothek erstellen", wenn Sie in der Umgebung bleiben möchten, nachdem die Quick-Bibliothek angelegt ist.
   - Wählen Sie die Befehlsfläche "Bibliothek erstellen und beenden", wenn Sie auf die DOS-Befehlsebene zurückkehren möchten, nachdem die Quick-Bibliothek angelegt ist.

Hinweis Wenn Sie eine Quick-Bibliothek erstellen, sollten Sie darauf achten, daß mindestens eines der Module in der Bibliothek eine Anweisung zur Ereignisverfolgung enthalten muß, falls die Quick-Bibliothek jemals mit einem Nicht-Bibliotheksmodul verwendet wird, das Ereignisverfolgung, wie zum Beispiel Tastenverfolgung, benötigt. Diese Anweisung kann so einfach sein wie **TIMER OFF**; ohne diese Anweisung werden Ereignisse in der Quick-Bibliothek jedoch nicht korrekt verfolgt.

8.4 Wie Sie Quick-Bibliotheken verwenden

Dieser Abschnitt erläutert, wie Quick-Bibliotheken geladen werden, wenn Sie QuickBASIC starten, und wie Sie sich den Inhalt einer Quick-Bibliothek ansehen können. Außerdem werden Informationen gegeben, an die Sie sich erinnern sollten, wenn Prozeduren innerhalb einer Quick-Bibliothek Gleitkomma-Arithmetik durchführen.

8.4.1 Wie Sie eine Quick-Bibliothek laden

Um eine Quick-Bibliothek zu laden, müssen Sie beim Starten von QuickBASIC den Namen der gewünschten Bibliothek mit der folgenden Syntax auf der Befehlszeile angeben:

qb [*Progname*] **/l** [*qBiblname*]

Falls Sie QuickBASIC mit der Option /l laden und den Namen einer Bibliothek (*qBiblname* in diesem Beispiel) angeben, lädt QuickBASIC die angegebene Datei und bringt Sie in die Programmierumgebung. Der Inhalt der Bibliothek ist jetzt für den Einsatz verfügbar. Falls Sie QuickBASIC mit der Option /l starten, aber keine Bibliothek angeben, lädt QuickBASIC die Bibliothek QB.QLB (siehe Abschnitt 8.5).

Sie können QuickBASIC auch mit der Option **/run**, gefolgt sowohl von einem Programmnamen (*Progname*) als auch der Option **/l**, starten. In diesem Fall lädt QuickBASIC sowohl das Programm als auch die angegebene Quick-Bibliothek und startet das Programm anschließend, ohne in der Programmierumgebung anzuhalten.

Hinweis Wenn Sie Quick-Bibliotheken dazu einsetzen, Programmodule zu repräsentieren, sollten Sie nicht vergessen, die .MAK-Datei zu aktualisieren, um diese laufend zu den Modulen des sich entwickelnden Programms angepaßt zu halten. (Dies geschieht mit dem Befehl **Datei entfernen** aus dem Menü **Datei.**) Falls die .MAK-Datei nicht aktuell ist, könnte sie QuickBASIC dazu veranlassen, ein Modul zu laden, das eine Prozedurdefinition mit demselben Namen wie eines der in der Quick-Bibliothek definierten Module enthält, was zu der Fehlermeldung `Doppelte Definition` führt.

Sie können nur jeweils eine Quick-Bibliothek laden. QuickBASIC sucht nach der Quick-Bibliothek an den folgenden drei Stellen:

1. An der Stelle, die, falls vorhanden, in dem Befehlszeilen-Argument angegeben wurde.
2. In dem aktuellen Verzeichnis.
3. In dem Pfad, der von der LIB-Umgebungsvariablen angegeben ist. (Informationen zu Umgebungsvariablen finden Sie in Ihrer DOS-Dokumentation.)

Beispiel

Der folgende Befehl startet QuickBASIC und führt das REPORT.BAS benannte Programm mit den Routinen aus der Bibliothek FIGS.QLB aus:

```
qb /run report.bas /l figs.qlb
```

8.4.2 Gleitkomma-Arithmetik in Quick-Bibliotheken

BASIC-Prozeduren innerhalb von Quick-Bibliotheken stellen mit dem BC-Befehlszeilen-Compiler kompilierten Code dar. Diese Prozeduren haben mit ausführbaren Dateien wichtige Eigenschaften gemeinsam. Zum Beispiel führen sowohl ausführbare Dateien als auch Quick-Bibliotheken Gleitkomma-Arithmetik schneller und mit einem höheren Grad an Genauigkeit durch, als wenn dieselben Berechnungen innerhalb der QuickBASIC-Umgebung durchgeführt werden. Eine Erklärung, warum und wie dies in ausführbaren Dateien geschieht, finden Sie in Abschnitt 6.2.4, "Gleitkomma-Arithmetik in ausführbaren Dateien". Weitere Informationen zur Verwendung von Vergleichsoperatoren und Ausdrücken mit Gleitkommawerten finden Sie in Kapitel 3, "Ausdrücke und Operatoren", im *BASIC-Befehlsverzeichnis.*

8.4.3 Wie Sie sich den Inhalt einer Quick-Bibliothek ansehen

Da eine Quick-Bibliothek im wesentlichen eine Binärdatei ist, können Sie deren Inhalt nicht einfach mit einem Texteditor betrachten, um herauszufinden, was sie enthält. Auf Ihren Originaldisketten befindet sich das Hilfsprogramm QLBDUMP.BAS, das es Ihnen ermöglicht, alle Prozeduren einer angegebenen Bibliothek aufzulisten. Folgen Sie diesen Schritten, um sich den Inhalt einer Quick-Bibliothek anzusehen:

1. Starten Sie QuickBASIC.
2. Laden und starten Sie QLBDUMP.BAS.
3. Geben Sie als Antwort auf die Anfrage den Namen der Quick-Bibliothek ein, die Sie untersuchen möchten. Sie müssen die Erweiterung .QLB nicht angeben, wenn Sie den Dateinamen eintippen; die Eingabe der Erweiterung schadet jedoch nicht.

   Falls die angegebene Datei existiert und eine Quick-Bibliothek ist, zeigt das Programm eine Liste aller in der Bibliothek verwendeten Symbolnamen an. In diesem Zusammenhang beziehen sich Symbolnamen auf die Prozedurnamen in der Bibliothek.

Ein kommentiertes Listing von QLBDUMP.BAS finden Sie in Kapitel 3, "Datei- und Geräte-E/A", in *Programmieren in BASIC: Ausgewählte Themen.*

8.5 Die mitgelieferte Bibliothek (QB.QLB)

Wenn Sie QuickBASIC mit der Option /l aufrufen, aber keinen Namen einer Quick-Bibliothek angeben, lädt QuickBASIC automatisch eine QB.QLB benannte Datei. Diese Datei befindet sich auf Originaldiskette 3. Die Datei enthält drei Routinen, die **INTERRUPT, INT86OLD** und **ABSOLUTE** heißen und Unterstützung von Software-Unterbrechungen für Systemaufrufe sowie Unterstützung für **CALL ABSOLUTE** bieten. Um die Routinen aus QB.QLB zu verwenden, müssen Sie diese (oder eine andere Bibliothek, in die solche Routinen eingebunden sind) auf der Befehlszeile angeben, wenn Sie QuickBASIC aufrufen. Falls Sie diese Routinen zusammen mit anderen Routinen laden möchten, die Sie in Bibliotheken abgelegt haben, sollten Sie eine Kopie der Bibliothek QB.QLB anlegen und diese als Basis zur Erstellung einer Bibliothek verwenden, die alle von Ihnen benötigten Routinen enthält. Eine Beschreibung, wie diese Routinen in einer **CALL**-Anweisung aufgerufen werden, finden Sie in der Beschreibung für die Anweisung **CALL** im *BASIC-Befehlsverzeichnis*.

8.6 Die Dateinamenerweiterung .QLB

Die Erweiterung .QLB ist nur eine passende Vereinbarung. Sie können jede beliebige Erweiterung oder auch keine Erweiterung für Ihre Quick-Bibliotheksdateien verwenden. Bei der Verarbeitung der Option /l *qBiblname* geht QuickBASIC jedoch davon aus, daß der aufgeführte Name *qBiblname* die Erweiterung .QLB hat, wenn keine andere Erweiterung angegeben ist. Falls Ihre Quick-Bibliothek keine Erweiterung hat, müssen Sie diese als *qBiblname.* angeben, oder QuickBASIC sucht nach einer Datei mit dem von Ihnen angegebenen Basisnamen sowie der Erweiterung .QLB. Ein den Basisnamen abschließender Punkt (.) zeigt QuickBASIC an, daß der Dateiname keine Erweiterung hat.

8.7 Wie Sie eine Bibliothek von der Befehlszeile aus aufbauen

Nachdem Sie eine Bibliothek innerhalb der QuickBASIC-Umgebung aufgebaut haben, werden Sie das Vorhandensein zusätzlicher Dateien mit den Erweiterungen .OBJ und .LIB feststellen. Beim Anlegen von Quick-Bibliotheken steuert QuickBASIC die Arbeit dreier anderer Programme, BC, LINK und LIB, und faßt anschließend das, was diese Programme erzeugen, sowohl in einer Quick-Bibliothek als auch in einer selbständigen (.LIB) Bibliothek zusammen.

Nachdem dieser Prozeß abgeschlossen ist, existiert eine Objekt-Datei (.OBJ) für jedes Modul in Ihrer Quick-Bibliothek und eine einzelne Bibliotheksdatei (.LIB), die für jede Objektdatei ein Objektmodul enthält. Die Dateien mit der Erweiterung .OBJ sind jetzt überflüssig und können gelöscht werden. Dateien mit der Erweiterung .LIB sind jedoch sehr wichtig und sollten erhalten bleiben. Diese parallelen Bibliotheken sind die Dateien, die QuickBASIC verwendet, um aus Ihren Programmen ausführbare Dateien zu erstellen.

Sie können die Programme LINK und LIB verwenden, um von der Befehlszeile aus mit einer Stapeldatei sowohl Quick-Bibliotheken als auch selbständige (.LIB) Bibliotheken zu erstellen. Falls Sie in QuickBASIC Routinen einsetzen möchten, die ursprünglich in anderen Sprachen geschrieben und kompiliert wurden, *müssen* Sie per Befehlszeile zunächst die anderssprachigen Routinen in einer Quick-Bibliothek ablegen. Sobald die anderssprachigen Routinen sich einmal in der Bibliothek befinden, können Sie Ihre BASIC-Module von der Befehlszeile aus oder aus der QuickBASIC-Umgebung heraus einbinden.

Wenn Sie ein professioneller Software-Entwickler sind, sollten Sie dem Kunden sowohl die Quick- als auch die selbständige (.LIB) Version Ihrer Bibliothek liefern. Ohne die .LIB-Bibliotheken werden Endnutzer nicht in der Lage sein, Ihre Bibliotheksroutinen in ausführbare Dateien einzubinden, die sie mit QuickBASIC erstellen.

Hinweis Wenn Sie eine Quick-Bibliothek unter Verwendung des Linkers erstellen, muß die BQLB40.LIB-Bibliothek immer nach dem dritten Komma auf der **link**-Befehlszeile oder als Antwort auf die Anfrage "Bibliothek" angegeben werden.

8.8 Wie Sie in einer Quick-Bibliothek Routinen aus anderen Sprachen verwenden

Um Routinen aus anderen Sprachen in einer Quick-Bibliothek abzulegen, müssen Sie mit vorkompilierten oder vorassemblierten Objektdateien beginnen, die die anderssprachigen Routinen enthalten, die Sie verwenden möchten. Mehrere andere Sprachen sind für diesen Zweck einsetzbar, einschließlich Microsoft C, Microsoft Macro Assembler, Microsoft Pascal, Microsoft FORTRAN sowie jede andere Sprache, die Objektdateien anlegt, die kompatibel zu der Microsoft-Sprachenfamilie sind.

Die folgenden Schritte sind typisch für das Anlegen einer Quick-Bibliothek, die Routinen aus anderen Sprachen enthält:

1. Nehmen Sie an, Sie beginnen mit drei Modulen, die in FORTRAN, C und Macro Assembler erstellt wurden. Der erste Schritt besteht darin, jedes Modul mit dem entsprechenden Sprachübersetzer zu kompilieren oder zu assemblieren. Dies erzeugt Objektdateien, die wir FOR.OBJ, C.OBJ sowie ASM.OBJ nennen können.

2. Anschließend binden Sie die Objektdateien mit einer besonderen Option von LINK, **/Q**, die den Linker anweist, eine Quick-Bibliothek anzulegen, wie in der folgenden Befehlszeile gezeigt:

   ```
   LINK /Q FOR.OBJ C.OBJ ASM.OBJ, GEMISCH.QLB,,BQLB40.LIB;
   ```

   Der Linker interpretiert den Eintrag, der den Namen der Objektdateien folgt, (in diesem Fall GEMISCH.QLB) als den Zieldateinamen, der Name, unter dem die gebundenen Module abgelegt werden. Daher ist der Name der Quick-Bibliothek in diesem Fall GEMISCH.QLB.

3. Erstellen Sie nun parallel dazu eine .LIB-Bibliothek mit denselben Objektdateien, die Sie gerade zum Aufbau der Quick-Bibliothek verwendet haben. In diesem Fall ist der erste Name, der dem LIB-Befehl folgt, der Name der .LIB-Zielbibliothek:

   ```
   LIB GEMISCH.LIB+FOR.OBJ+C.OBJ+ASM.OBJ;
   ```

Wichtig Schritt 3 kann leicht übersehen werden, wenn Sie eine Bibliothek aufbauen, die anderssprachige Routinen enthält; dieser Schritt ist jedoch entscheidend, wenn Sie die Bibliothek dazu verwenden möchten, eine selbständig ausführbare Datei zu erzeugen. Ohne diese parallelen selbständigen (.LIB) Bibliotheken kann QuickBASIC keine ausführbare Datei erstellen, die Routinen der Bibliotheken enthält.

4. Mit den anderssprachigen Routinen, jetzt in einer Quick-Bibliothek, sowie den ursprünglichen Objektdateien in einer selbständigen Bibliothek, die denselben Basisnamen hat, können Sie in die QuickBASIC-Umgebung zurückkehren und so viele BASIC-Module in die Bibliothek einbinden, wie es der verfügbare Speicherplatz zuläßt.

Eine umfassende Beschreibung von LINK und LIB finden Sie in Kapitel 9," Wie Sie aus DOS heraus kompilieren und binden". Informationen zum Schreiben anderssprachiger Routinen sowie zu BASIC-Prozeduren, die gegenseitig aufrufbar sind, finden Sie in Anhang C, "Wie Sie C- und Assembler-Routinen aufrufen".

Hinweis QuickBASIC bietet eine BASIC-Systemebenen-Funktion (**B_OnExit**). Diese Funktion kann von anderssprachigen Routinen aufgerufen werden, um eine Beendigungsprozedur anzumelden, die aufgerufen wird, wenn ein BASIC-Programm endet oder erneut gestartet wird, sofern eine Quick-Bibliothek vorhanden ist. Eine Beschreibung dieser Routine finden Sie in Abschnitt C.9.

8.9 Quick-Bibliotheken und Speicherplatz

Da eine Quick-Bibliothek im wesentlichen eine ausführbare Datei ist (obwohl sie nicht selbst von der DOS-Befehlszeile aufgerufen werden kann), ist sie ziemlich groß im Vergleich zur Summe ihrer Quelldateien. Dies setzt eine Obergrenze für die Anzahl an Routinen, die Sie in einer Quick-Bibliothek ablegen können. Um zu bestimmen, wie groß Ihre Quick-Bibliothek werden kann, addieren Sie den Speicherplatz, den DOS, QB.EXE und das Hauptmodul Ihres Programmes benötigen. Eine einfache Methode, diese Faktoren zu berechnen, besteht darin, Ihre Maschine zu booten, Ihr Programm mit QuickBASIC zu starten und diesen Befehl in das Direkt-Fenster einzugeben:

```
print fre (-1)
```

Dieser Befehl zeigt Ihnen die Anzahl freier Speicherbytes an und gibt Ihnen damit die maximale Größe für eine mit diesem Programm verknüpfte Quick-Bibliothek an. In den meisten Fällen ist der für eine Quick-Bibliothek erforderliche Speicherplatz ungefähr gleich der Größe Ihrer Diskettendatei. Eine Ausnahme zu dieser Faustregel stellt eine Bibliothek mit Prozeduren dar, die viele Zeichenketten verwenden und möglicherweise mehr Speicherplatz benötigen.

8.10 Wie Sie kompakte ausführbare Dateien erstellen

Wie oben erläutert, legt QuickBASIC, wenn es eine Quick-Bibliothek anlegt, auch eine selbständige (.LIB) Bibliothek mit Objektmodulen an, in der jedes Objektmodul mit einem der Module in der Quick-Bibliothek korrespondiert. Wenn Sie eine ausführbare Datei erstellen, dann durchsucht QuickBASIC die selbständige .LIB-Datei nach Objektmodulen, die die Prozeduren enthalten, auf die in dem Programm Bezug genommen wird.

Falls ein Objektmodul keine der in dem Programm aufgeführten Prozeduren enthält, wird es nicht in die ausführbare Datei eingebunden. Ein einzelnes Modul kann jedoch viele Prozeduren enthalten, und selbst wenn von diesen nur eine in dem Programm erscheint, werden alle Prozeduren in das Programm aufgenommen. Selbst wenn ein Programm aus einem bestimmten Modul nur eine von vier Prozeduren verwendet, wird das gesamte Modul Teil der ausführbaren Datei.

Um Ihre ausführbaren Dateien so kompakt wie möglich werden zu lassen, sollten Sie eine "knappe" Bibliothek aufbauen, das heißt, eine Bibliothek, in der jedes Modul nur Prozeduren enthält, die direkt miteinander verknüpft sind. Falls Sie irgendwelche Zweifel darüber haben, was eine Bibliothek enthält, können Sie deren Inhalt mit dem in Abschnitt 8.4.3, "Wie Sie sich den Inhalt einer Quick-Bibliothek ansehen", beschriebenen Hilfsprogramm QLBDUMP.BAS auflisten.

9 Wie Sie aus DOS heraus kompilieren und binden

Dieses Kapitel erläutert, wie Sie außerhalb der QuickBASIC-Umgebung kompilieren und binden können. Wenn Sie dieses Kapitel gelesen haben, werden Sie wissen, wie Sie

- Von der DOS-Befehlszeile aus mit dem **bc**-Befehl kompilieren.
- Ausführbare Dateien anlegen und Programm-Objektdateien mit dem Befehl **link** binden.
- Selbständige (.LIB) Bibliotheken mit dem Befehl **lib** erstellen und pflegen.

9.1 BC, LINK und LIB

Zusätzlich zu QB.EXE enthält das Microsoft QuickBASIC-Paket verschiedene Programme, die für Sie unter bestimmten Bedingungen hilfreich sind:

Programm	***Funktion***
BC.EXE	Wenn Sie den Befehl **EXE-Datei erstellen** oder den Befehl **Bibliothek erstellen** aus dem Menü **Ausführen** wählen, ruft QuickBASIC den BASIC-Befehlszeilen-Compiler (BC) auf, um Zwischendateien zu erstellen, die als Objektdateien bezeichnet werden. Diese werden zusammengebunden, um Ihr Programm oder Ihre Quick-Bibliothek zu bilden. BC steht Ihnen auch jederzeit zur Verfügung, wenn Sie Programme außerhalb der QuickBASIC-Umgebung kompilieren möchten. Sie bevorzugen vielleicht BC, wenn Sie ein Programm nur kompilieren möchten, das Sie mit einem anderen Texteditor geschrieben haben. Sie *müssen* BC jedoch nur dann von der Befehlszeile aus verwenden, wenn Ihr Programm zu groß ist, um innerhalb der QuickBASIC-Umgebung im Speicher kompiliert zu werden, oder wenn Sie es wünschen, daß Ihre ausführbare Datei kompatibel zu dem CodeView-Debugger ist.
LINK.EXE	QuickBASIC verwendet den Microsoft Overlay Linker (LINK), um von BC erstellte Objektdateien mit den entsprechenden Bibliotheken zu binden und eine ausführbare Datei zu erzeugen. Genauso wie BC können Sie LINK immer dann verwenden, wenn Sie Objektdateien binden oder Quick-Bibliotheken erstellen möchten.
LIB.EXE	Sie können den Microsoft-Bibliotheksmanager (LIB) verwenden, um aus den von BC erzeugten Objektdateien selbständige Bibliotheken zu erstellen. QuickBASIC selbst verwendet LIB, um solche Bibliotheken zu erstellen, und verwendet diese anschließend, wenn Sie den Befehl **EXE-Datei erstellen** aus dem Menü **Ausführen** wählen.

9.2 Vorteile der Verwendung von BC

Es folgen einige Gründe zur Verwendung von BC, um außerhalb der QuickBASIC-Umgebung zu kompilieren und zu binden:

- Um einen anderen Texteditor zu verwenden.
- Um ausführbare Programme zu erzeugen, die mit dem Microsoft CodeView-Debugger debugged werden können.
- Um Listendateien zu erstellen, die für das Debuggen in selbständig ausführbaren Programmen verwendet werden.
- Um Optionen zu verwenden, die nicht innerhalb der QuickBASIC-Umgebung verfügbar sind, wie zum Beispiel zeilenweises Speichern von Datenfeldern.
- Um mit NOCOM.OBJ (eine Datei, die mit QuickBASIC geliefert wird) zu binden, die die Größe von ausführbaren Dateien in Programmen, die keine **COM**-Anweisungen verwenden, reduziert.

9.3 Der Prozeß des Kompilierens und Bindens

Um außerhalb der QuickBASIC-Umgebung von einer BASIC-Quelldatei ein selbständiges Programm zu erstellen, folgen Sie diesen Schritten:

1. Kompilieren Sie jede Quelldatei, um eine Objektdatei zu erstellen.
2. Binden Sie die Objektdateien mit einer oder mehreren selbständigen Bibliotheken, um eine ausführbare Datei anzulegen. Bevor eine ausführbare Datei angelegt wird, stellt der Linker sicher, daß alle Prozeduraufrufe in den Quelldateien mit den Prozeduren in den Bibliotheken oder mit Prozeduren in anderen Objektdateien übereinstimmen.

Sie können eine der beiden folgenden Methoden zum Kompilieren und Binden verwenden:

1. Sie können in einzelnen Schritten kompilieren und binden, indem Sie die Befehle **bc** und **link** verwenden.
2. Sie können eine Stapeldatei anlegen, die alle Kompilier- und Bindebefehle enthält. Diese Methode ist besonders hilfreich, wenn Sie beim Kompilieren und Binden Ihrer Programme immer dieselben Optionen verwenden.

Hinweis Wenn QuickBASIC Ihr Programm innerhalb der Umgebung kompiliert und bindet, wird die Linker-Option /E automatisch gesetzt. Wenn Sie jedoch den Befehl **link** außerhalb der QuickBASIC-Umgebung einsetzen, müssen Sie die Option /E ausdrücklich angeben, um die Größe der ausführbaren Datei zu minimieren sowie die Ladegeschwindigkeit für das Programm zu maximieren.

Die Abschnitte 9.4 und 9.5 erläutern, wie in einzelnen Schritten kompiliert und gebunden wird.

9.4 Wie Sie mit dem Befehl bc kompilieren

Sie können mit dem Befehl **bc** auf eine der folgenden beiden Arten kompilieren:

1. Geben Sie alle Informationen auf einer einzelnen Befehlszeile ein, verwenden Sie die folgende Syntax:

   bc *Quelldatei* [,[*Objektdatei*] [,[*Listendatei*]]] [*Optionsliste*][;]

2. Geben Sie `bc` ein und antworten Sie auf die folgenden Anfragen:

   ```
   Quelldatei [.BAS]:
   Objektdatei [Basisname.OBJ]:
   Quell-Listing: [NUL.LST]:
   ```

Die folgende Tabelle zeigt die Eingabe, die Sie auf der **bc**-Befehlszeile oder als Antwort auf jede Anfrage vornehmen müssen:

Feld	***Anfrage***	***Eingabe***
Quelldatei	"Quelldatei"	Der Name Ihrer Quelldatei
Objektdatei	"Objektdatei"	Der Name der Objektdatei, die Sie erzeugen
Listendatei	"Quell-Listing"	Der Name der Datei, die ein Quell-Listing enthält. Die Listendatei enthält die Adresse jeder Zeile Ihrer Quelldatei, den Text der Quelldatei, deren Größe und jede Fehlermeldung, die während der Kompilierung erzeugt wurde.
Optionsliste	Bietet Optionen nach jeder Anfrage	Eine der in Abschnitt 9.4.2, "Wie Sie die bc-Befehlsoptionen verwenden", beschriebene Compiler-Optionen.

9.4.1 Wie Sie Dateinamen angeben

Der **bc**-Befehl geht, basierend auf den Pfadnamen und Erweiterungen, die Sie für die Dateien verwenden, von bestimmten Voraussetzungen hinsichtlich der von Ihnen angegebenen Dateien aus. Der folgende Abschnitt beschreibt diese Voraussetzungen und andere Regeln, denen Sie bei der Angabe von Dateinamen für den Befehl **bc** folgen sollten.

9.4.1.1 Groß- und Kleinbuchstaben

Sie können für Dateinamen jede beliebige Kombination von Groß- und Kleinbuchstaben verwenden.

Beispiel

Der Befehl **bc** betrachtet die folgenden drei Dateinamen als identisch:

```
abcde.BAS
ABCDE.BAS
aBcDe.Bas
```

9.4.1.2 Erweiterungen für Dateinamen

Ein DOS-Dateiname besteht aus zwei Teilen: dem "Basisnamen", der aus allen Zeichen bis zu dem Punkt (.) besteht (ohne diesen zu enthalten), sowie der "Erweiterung", die den Punkt und bis zu drei Zeichen, die dem Punkt folgen, umfaßt. Im allgemeinen kennzeichnet die Erweiterung den Typ der Datei (zum Beispiel, ob die Datei eine BASIC-Quelldatei, eine Objektdatei, eine ausführbare Datei oder eine selbständige Bibliothek ist).

BC und LINK verwenden für Dateinamen die in der folgenden Liste erläuterten Erweiterungen:

Erweiterung	*Erklärung*
.BAS	BASIC-Quelldatei
.OBJ	Objektdatei
.LIB	selbständige Bibliotheksdatei
.LST	Die von BC erstellte Listendatei
.MAP	Datei für Symbole des gebundenen Programms
.EXE	Ausführbare Datei

9.4.1.3 Pfadnamen

Jeder Dateiname kann einen vollständigen oder teilweisen Pfadnamen enthalten. Ein vollständiger Pfadname beginnt mit der Laufwerksangabe; ein Teilpfadname besteht aus einem oder mehreren Verzeichnisnamen vor dem Dateinamen, enthält jedoch keine Laufwerksangabe.

Ein vollständiger Pfadname erlaubt es Ihnen, Dateien mit unterschiedlichen Pfaden als Eingabe für den Befehl **bc** anzugeben, und ermöglicht es Ihnen, Dateien auf unterschiedlichen Laufwerken oder in unterschiedlichen Verzeichnissen des aktuellen Laufwerks zu erzeugen.

Hinweis Für Dateien, die Sie mit BC anlegen, können Sie einen Pfadnamen angeben, der mit einem rückwärtigen Schrägstrich (\) endet, um die Datei in diesem Pfad anzulegen. Wenn BC die Datei anlegt, verwendet es den Standardnamen für die Datei.

9.4.2 Wie Sie die bc-Befehlsoptionen verwenden

Optionen des Befehls **bc** bestehen entweder aus einem Schrägstrich (/) oder aus einem Trennungsstrich (-), denen ein oder mehrere Buchstaben folgen. (In diesem Handbuch werden für Optionen Schrägstriche verwendet.)

Die **bc**-Befehlszeilen-Optionen werden in der folgenden Liste erläutert:

Option	*Beschreibung*
/a	Erzeugt ein Listing mit disassembliertem Objektcode für jede Quellzeile und zeigt den Assembler-Code, der vom Compiler erzeugt wird.
/ah	Ermöglicht es dynamischen Datenfeldverbunden, Zeichenketten fester Länge und numerischen Daten, den gesamten verfügbaren Speicher einzunehmen. Wenn diese Option nicht angegeben ist, beträgt die maximale Größe pro Datenfeld 64K. Beachten Sie, daß diese Option keine Auswirkungen darauf hat, wie Datengrößen an Prozeduren übergeben werden.
/**c**:*Puffergröße*	Setzt für jeden Kommunikationsanschluß die Größe des Puffers, der Ferndaten über einen asynchronen Kommunikationsadapter empfängt. (Dem Übertragungspuffer wird für jeden Übertragungsanschluß eine Kapazität von 128 Bytes zugewiesen; der Puffer kann nicht von der **bc**-Befehlszeile aus verändert werden.) Diese Option hat keine Auswirkungen, falls keine asynchrone Datenübertragungskarte vorhanden ist. Die Standard-Puffergröße beträgt für beide Anschlüsse insgesamt 256 Bytes; die maximale Größe beträgt 32.767 Bytes.

Option	*Beschreibung*
/d	Erzeugt Debug-Code für Laufzeit-Fehlerprüfung und schaltet STRG+UNTBR ein. Diese Option ist identisch mit der Option "Debug-Code erstellen" des Befehls **EXE-Datei erstellen** aus dem Menü **Ausführen** innerhalb der Umgebung.
/e	Zeigt das Vorhandensein von **ON ERROR**- mit **RESUME** *Zeilennummer*-Anweisungen an. (Siehe auch die Erläuterung der Option /x in dieser Liste.)
/mbf	Bewirkt, daß die QuickBASIC-Konvertierungsfunktionen Zahlen im IEEE-Format als Zahlen im Microsoft-Binär-Format behandeln. Die eingebauten Funktionen **MKS$**, **MKD$**, **CVS** und **CVD** werden jeweils übertragen in **MKSMBF$**, **MKDMBF$**, **CVSMBF** und **CVDMBF**. Weitere Informationen zu diesen Funktionen finden Sie im *BASIC-Befehlsverzeichnis*.
/o	Ersetzt BRUN40.LIB durch die Laufzeit-Bibliothek BCOM40.LIB. Weitere Informationen zur Verwendung von Bibliotheken finden Sie in Abschnitt 6.2, "Wie Sie innerhalb von QuickBASIC ausführbare Dateien erstellen".
/r	Speichert Datenfelder zeilenweise. Normalerweise speichert BASIC Datenfelder spaltenweise. Diese Option ist hilfreich, falls Sie anderssprachige Routinen verwenden, die Datenfelder zeilenweise speichern.
/s	Schreibt in Anführungszeichen gesetzte Zeichenketten in die Objektdatei, und nicht in die Symboltabelle. Verwenden Sie diese Option, wenn die Fehlermeldung `Speicherkapazität reicht nicht aus` in einem Programm auftritt, in dem viele Zeichenketten-konstanten vorkommen.
/v	Schaltet Ereignisverfolgung für Datenübertragung (**COM**), Lichtstift (**PEN**), Joystick (**STRIG**), Uhr (**TIMER**), Musikpuffer (**PLAY**) und Funktionstasten (**KEY**) ein. Verwenden Sie diese Option, damit zwischen jeder Anweisung auf das Auftreten eines Ereignisses geprüft wird.
/w	Schaltet die Ereignisverfolgung für dieselben Anweisungen wie /v ein. Es prüft jedoch zwischen jeder Zeile auf das Auftreten eines Ereignisses.
/x	Kennzeichnet das Vorhandensein von **ON ERROR** mit **RESUME**, **RESUME NEXT** oder **RESUME 0**.

Option	*Beschreibung*
/zd	Erzeugt eine Objektdatei, die Sätze mit Zeilennummern enthält, die den Zeilennummern der Quelldatei entsprechen. Diese Option ist hilfreich, wenn Sie mit dem Microsoft "Symbolic Debug Utility" (SYMDEB), der mit dem Microsoft Macro Assembler Version 4.0 verfügbar ist, Debuggen auf Quellebene durchführen möchten.
/zi	Erzeugt eine Objektdatei, die Debug-Informationen enthält, die der Microsoft CodeView-Debugger verwendet, der mit Microsoft C Version 5.0 und dem Microsoft Macro Assembler Version 5.0 verfügbar ist.

9.5 Binden

Nachdem Sie Ihr Programm kompiliert haben, müssen Sie die Objektdatei mit den passenden Bibliotheken binden, um ein ausführbares Programm zu erzeugen. Sie können den **link**-Befehl auf eine der folgenden Arten verwenden:

- Geben Sie auf der Befehlszeile folgendes ein:

  link *Objdatei* [,[*Exedatei*][,[*Mapdatei*][,[*Bibl*]]][*Linkoptionen*][;]]

  Die Befehlszeile darf nicht mehr als 128 Zeichen enthalten.

- Schreiben Sie `link` und antworten Sie auf die folgenden Anfragen:

  ```
  Objektmodule [.OBJ]:
  Ausführbare Datei [Basisname.EXE]:
  List-Datei [NUL.MAP]:
  Bibliotheken [.LIB]:
  ```

  Um nach einer Anfrage mehrere Dateien anzugeben, geben Sie am Ende der Zeile ein Pluszeichen (+) ein. Die Anfrage erscheint erneut auf der nächsten Zeile, und Sie können mit der Eingabe für die Anfrage fortfahren.

- Richten Sie eine Datei (als "Antwortdatei" bezeichnet) ein, die Antworten auf die **link**-Befehlsanfragen enthält, und geben Sie anschließend einen **link**-Befehl folgender Form ein:

  link @ *Dateiname*

  An dieser Stelle ist *Dateiname* der Name der Antwortdatei. Jeder Antwort können Sie Linker-Optionen hinzufügen oder auf einer oder mehreren separaten Zeilen Optionen angeben. Die Antworten müssen in derselben Reihenfolge vorliegen wie die oben erläuterten Befehlsanfragen. Sie können auch den Namen einer Antwortdatei nach jeder Linker-Anfrage oder an beliebiger Stelle auf der **link**-Befehlszeile eingeben.

Die folgende Tabelle zeigt die Eingaben, die Sie auf der **link**-Befehlszeile oder als Anwort auf jede Anfrage angeben müssen.

Feld	***Anfrage***	***Eingabe***
Objdatei	"Objektmodule"	Eine oder mehrere Objektdateien, die Sie binden. Die Objektdateien werden durch Pluszeichen oder Leerzeichen voneinander getrennt.
Exedatei	"Ausführbare" Datei"	Name der ausführbaren Datei, die Sie erstellen, wenn Sie dieser einen Namen oder eine Erweiterung geben, die vom Standard abweichen. Sie sollten immer die Erweiterung .EXE verwenden, da DOS erwartet, daß ausführbare Dateien diese Erweiterung haben.
Mapdatei	"List-Datei"	Name der Datei, die eine Symboltabelle enthält, falls Sie eine solche anlegen.* Sie können auch einen der folgenden DOS-Gerätenamen angeben, um die Map-Datei auf dieses Gerät umzuleiten: AUX für ein Zusatzgerät, CON für die Konsole (Terminal), PRN für einen Drucker oder NUL für kein Gerät (so daß keine Map-Datei angelegt wird). Ein Beispiel zu einer Map-Datei sowie Informationen über deren Inhalt finden Sie in Abschnitt 9.5.5.10.

Feld	*Anfrage*	*Eingabe*
Bibl	"Bibliotheken"	Eine oder mehrere selbständige Bibliotheken (oder Verzeichnisse, die nach selbständigen Bibliotheken durchsucht werden sollen), getrennt durch Pluszeichen oder Leerzeichen. Die Anfrage "Bibliotheken" ermöglicht es Ihnen, bis zu 16 Bibliotheken anzugeben; jede weitere Bibliothek wird ignoriert. In Abschnitt 9.5.3 finden Sie die Regeln, mit denen dem Linker Bibliotheksnamen angegeben werden.
Linkoptionen	Gibt Optionen nach jeder Antwort	Eine beliebige der in den Abschnitten 9.5.5.1 bis 9.5.5.11 beschriebenen Linker-Optionen. Sie können Linker-Optionen auf der Befehlszeile an beliebiger Stelle angeben.

* Eine andere Möglichkeit, eine Map-Datei zu erstellen, besteht darin, für den Befehl link die Option /MAP anzugeben (siehe Abschnitt 9.5.5.10).

Falls Sie eine Antwortdatei verwenden, muß jede Antwort der in der obigen Tabelle beschriebenen Regeln folgen.

9.5.1 Vorgabewerte des Linkers

Sie können Vorgabewerte für alle vom Linker benötigten Informationen auf eine der folgenden drei Arten wählen:

1. Um den Vorgabewert für einen Befehlszeileneintrag zu wählen, lassen Sie den bzw. die Dateinamen vor dem Eintrag aus und geben nur das erforderliche Komma ein. Die einzige Ausnahme hierzu ist der Vorgabewert für den Eintrag der *Mapdatei*: Wenn Sie ein Komma als Platzhalter für diesen Eintrag verwenden, erzeugt der Linker eine Map-Datei.
2. Um den Vorgabewert für eine Anfrage zu wählen, betätigen Sie einfach die EINGABETASTE.
3. Um die Vorgabewerte für alle verbleibenden Befehlszeileneinträge oder -anfragen zu wählen, geben Sie nach jedem Eintrag bzw. nach jeder Anfrage ein Semikolon ein. Die einzigen erforderlichen Angaben sind ein oder mehrere Namen von Objektdateien.

Die folgende Liste zeigt die Vorgabewerte, die der Linker für ausführbare Dateien, Map-Dateien sowie Bibliotheken verwendet:

Dateityp	***Vorgabewert***
Ausführbar	Der Basisname der ersten angegebenen Objektdatei und die Erweiterung .EXE. Um die ausführbare Datei umzubenennen, müssen Sie nur den neuen Basisnamen angeben; wenn Sie einen Dateinamen ohne Erweiterung angeben, fügt der Linker automatisch die Erweiterung .EXE an.
Map	Der besondere Dateiname NUL.MAP, der dem Linker mitteilt, *keine* Map-Datei zu erzeugen. Um eine Map-Datei anzulegen, müssen Sie nur den Basisnamen angeben; wenn Sie einen Dateinamen ohne Erweiterung angeben, fügt der Linker automatisch die Erweiterung .MAP an.
Bibliotheken	Bibliotheken, die in den angegebenen Objektdateien benannt sind. Die Standard-Bibliothek ist BRUN40.LIB. Falls Sie eine andere als die Standard-Bibliothek angeben, müssen Sie nur den Basisnamen eingeben; wenn Sie einen Bibliotheksnamen ohne Erweiterung eingeben, fügt der Linker automatisch die Erweiterung .LIB an. Informationen darüber, wie andere Bibliotheken als die Standard-Bibliotheken angegeben werden, finden Sie in Kapitel 9.5.3.

Hinweis Wenn Sie eine selbständig ausführbare Datei binden und Ihr Programm keine **COM**-Anweisungen verwendet, wird es etwa 4K kleiner sein, wenn Sie es mit NOCOM.OBJ binden. (Eine Datei, die mit QuickBASIC geliefert wird.)

Beispiele

Das folgende Beispiel zeigt eine Antwortdatei. Sie weist den Linker an, die vier Objektmodule RAHMEN, TEXT, TABELLE und SKIZZE zu binden. Es werden sowohl die als RAHMEN.EXE benannte ausführbare Datei als auch die als RAHMENSY.MAP benannte Map-Datei erzeugt. Die Option **/PAUSE** veranlaßt den Linker anzuhalten, bevor die ausführbare Datei erzeugt wird. Dies ermöglicht einen eventuell notwendigen Diskettenwechsel. Die Option **/MAP** teilt dem Linker mit, globale Symbole und Adressen in die Map-Datei einzufügen. Der Linker bindet außerdem jede aus der Bibliotheksdatei GRAF.LIB benötigte Routine. Weitere Informationen zu den Optionen **/PAUSE** und **/MAP** finden Sie in den Abschnitten 9.5.5.2 und 9.5.5.10.

```
RAHMEN TEXT TABELLE SKIZZE
/PAUSE /MAP
RAHMENSY
GRAF.LIB
```

Im folgenden Beispiel lädt und bindet der Linker die Objektdateien RAHMEN.OBJ, TEXT.OBJ, TABELLE.OBJ und SKIZZE.OBJ und durchsucht die Bibliotheksdatei HILFLIB.LIB nach ungelösten Bezügen. Die ausführbare Datei wird standardmäßig RAHMEN.EXE genannt. Außerdem wird eine Map-Datei erzeugt, die RAHMENSY.MAP heißt.

```
LINK RAHMEN+TEXT+TABELLE+SKIZZE, ,RAHMENSY, HILFLIB.LIB
```

Das folgende Beispiel verdeutlicht, wie eine Anfrage mit der Eingabe eines Pluszeichens (+) am Ende Ihrer Antwort fortgesetzt wird. Das Beispiel bindet alle angegebenen Objektdateien und erstellt anschließend eine ausführbare Datei. Da als Antwort auf die Anfrage "Ausführbare Datei" ein Semikolon eingegeben wird, wird der ausführbaren Datei der Standardname gegeben: der Basisname der ersten angegebenen Objektdatei (RAHMEN) mit der Erweiterung .EXE. Die Vorgabewerte werden auch für die verbleibenden Anfragen verwendet; daher wird keine Map-Datei angelegt, und für das Binden werden die in den Objektdateien benannten Standard-Bibliotheken verwendet.

```
LINK
Objektmodule [.OBJ]: RAHMEN TEXT TABELLE SKIZZE+
Objektmodule [.OBJ]: BASIS ZURUECK SPALNUM+
Objektmodule [.OBJ]: ZEILNUM
Ausführbare Datei [RAHMEN.EXE]: ;
```

9.5.2 Wie Sie dem Linker Dateien angeben

Die Regeln zur Angabe von Dateinamen für den Linker sind dieselben wie die Regeln zur Angabe von Dateinamen für den Befehl **bc**: Groß- und Kleinbuchstaben können beliebig verwendet werden, und Dateinamen können Pfadnamen enthalten, die dem Linker mitteilen, in dem angegebenen Pfad nach Dateien zu suchen bzw. Dateien anzulegen. Weitere Informationen finden Sie in Abschnitt 9.4.1.

9.5.3 Wie Sie dem Linker Bibliotheken angeben

Normalerweise müssen Sie dem Linker den Namen einer selbständigen Bibliothek nicht angeben. Wenn der **bc**-Befehl Objektdateien erzeugt, plaziert dieser in jeder Objektdatei den Namen der richtigen selbständigen Bibliothek für diese Objektdatei. Wenn die Objektdatei dem Linker übergeben wird, sucht der Linker nach einer Bibliothek mit dem Namen, der in der Objektdatei steht, und bindet die Objektdatei automatisch mit dieser Bibliothek.

Um Objektdateien mit einer anderen selbständigen Bibliothek als einer Standard-Bibliothek zu binden, teilen Sie dem Linker den Namen der Nicht-Standard-Bibliothek mit. Sie können den Bibliotheksnamen auf eine der folgenden Arten eingeben:

- Nach dem dritten Komma auf der **link**-Befehlszeile. Kommata folgen der Liste von Objektdateinamen, dem Namen der ausführbaren Datei sowie dem Namen der List-Datei. Der letzte Name entspricht dem Bibliotheksnamen.
- Als Antwort auf die Anfrage "Bibliotheken" des **link**-Befehls.

Bevor er die Standard-Bibliotheken durchsucht, durchsucht der Linker die von Ihnen angegebenen Bibliotheken, um externe Bezüge aufzulösen.

Aus einem der folgenden Gründe möchten Sie vielleicht mit einer anderen selbständigen Bibliothek als der Standard-Bibliothek binden:

- Binden mit zusätzlichen selbständigen Bibliotheken.
- Binden mit Bibliotheken in unterschiedlichen Pfaden. Wenn Sie für die Bibliothek einen kompletten Pfadnamen angeben, sucht der Linker nur in diesem Pfad nach der Bibliothek. Andernfalls sucht er an einer der folgenden drei Stellen:

  1. In dem aktuellen Arbeitsverzeichnis.
  2. In jedem Pfad oder Laufwerk, den/das Sie hinter dem dritten Komma auf der Befehlszeile angeben.
  3. An den Stellen, die von der LIB-Umgebungsvariablen angegeben werden.

- Ignorieren der in der Objektdatei benannten Bibliothek. In diesem Fall müssen Sie die Linker-Option **/NOD** zusätzlich zu der Bibliothek angeben, die Sie zum Binden benutzen möchten. Weitere Informationen zu der Option **/NOD** finden Sie in Abschnitt 9.5.5.8.

9.5.4 Speicheranforderungen des Linkers

Der Linker verwendet für den Bindevorgang den verfügbaren Speicher. Falls die zu bindenden Dateien eine Ausgabedatei bilden, die den verfügbaren Speicherbereich überschreitet, legt der Linker eine als Speicher dienende temporäre Diskettendatei an. Diese temporäre Datei wird, abhängig von der DOS-Version, auf eine der folgenden Arten behandelt:

- Der Linker verwendet das von der DOS-Umgebungsvariablen TMP angegebene Verzeichnis, um eine temporäre Datei zu erstellen. Falls die Variable TMP zum Beispiel auf C:\TEMPDIR gesetzt wurde, würde der Linker die temporäre Datei in C:\TEMPDIR schreiben.

  Wenn keine TMP-Umgebungsvariable vorhanden ist, oder das von TMP angegebene Verzeichnis nicht existiert, schreibt der Linker die temporäre Datei in das aktuelle Arbeitsverzeichnis.

- Wenn der Linker mit DOS-Version 3.0 oder größer läuft, verwendet er einen DOS-Systemaufruf, um eine temporäre Datei mit einem in dem Verzeichnis der temporären Datei einmaligen Namen zu erzeugen.
- Wenn der Linker mit einer DOS-Version kleiner 3.0 läuft, erzeugt er eine VM.TMP benannte temporäre Datei.

Wenn der Linker eine temporäre Diskettendatei erstellt, sehen Sie die Meldung

```
Temporäre Datei Tempdatei ist angelegt.
Diskette in Laufwerk Buchstabe nicht wechseln
```

wobei *Tempdatei* ".\" gefolgt von entweder dem Dateinamen VM.TMP oder einem von DOS erzeugten Namen und *Buchstabe* das Laufwerk, das die temporäre Datei enthält, ist.

Die Meldung

```
Diskette in Laufwerk Buchstabe nicht wechseln
```

erscheint nur dann, wenn das mit *Buchstabe* bezeichnete Laufwerk eine Diskettenstation ist. Falls diese Meldung erscheint, entfernen Sie die Diskette solange nicht aus dem Laufwerk, bis das Binden beendet ist. Wenn Sie die Diskette entfernen, verhält sich der Linker unvorhersehbar, und Sie könnten die folgende Meldung sehen:

```
Unerwartetes Dateiende in temporärer Datei
```

Wenn Sie diese Meldung sehen, beginnen Sie abermals mit dem Binden.

Die temporäre Datei, die der Linker anlegt, ist nur eine Arbeitsdatei. Nach Abschluß des Bindens wird sie von dem Linker gelöscht.

Hinweis Geben Sie keiner Ihrer eigenen Dateien den Namen VM.TMP. Der Linker gibt eine Fehlermeldung aus, wenn er eine existierende Datei dieses Namens findet.

9.5.5 Wie Sie die Linker-Optionen verwenden

Alle Linker-Optionen beginnen mit dem Zeichen der Linker-Option, dem Schrägstrich (/). Die Schreibweise spielt für Linker-Optionen keine Rolle; zum Beispiel sind /NOI und /noi identisch.

Sie können Linker-Optionen abkürzen, um sich Mühe und Platz zu sparen. Die kürzeste gültige Abkürzung für jede Option wird von der Syntax der Option vorgegeben. Zum Beispiel beginnen mehrere Optionen mit den Buchstaben "NO"; daher müssen Abkürzungen für diese Optionen länger als "NO" sein, um eindeutig zu sein. "NO" können Sie nicht als Abkürzung für die Option **/NOIGNORECASE** verwenden, da der Linker nicht unterscheiden kann, welche der mit "NO" beginnenden Optionen Sie wünschen. Die kürzeste gültige Abkürzung für diese Option lautet **/NOI.**

Abkürzungen müssen mit dem ersten Buchstaben der Option beginnen und müssen zusammenhängend bis zum letzten eingegebenen Buchstaben sein. Auslassungen und Änderungen der Buchstabenreihenfolge sind nicht erlaubt.

Einige Linker-Optionen arbeiten mit numerischen Argumenten. Ein numerisches Argument kann wie folgt aussehen:

- Eine Dezimalzahl von 0 bis 65.535.
- Eine oktale Zahl von 0 bis 0177777. Eine Zahl wird als oktal interpretiert, wenn sie mit einer Null (0) beginnt. Die Zahl `10` ist zum Beispiel eine Dezimalzahl, aber die Zahl `010` ist eine Oktalzahl, die gleich dezimal 8 ist.
- Eine hexadezimale Zahl von 0 bis 0xFFFF. Eine Zahl wird als hexadezimale Zahl interpretiert, wenn sie mit einer von einem x oder X gefolgten 0 beginnt. Zum Beispiel ist `0x10` eine hexadezimale Zahl, die gleich dezimal 16 ist.

Unabhängig davon, wo die Optionen angegeben werden, wirken sich Linker-Optionen auf alle Dateien des Bindevorganges aus.

Wenn Sie beim Binden normalerweise immer dieselben Linker-Optionen verwenden, dann können Sie die Umgebungsvariable LINK dazu einsetzen, bei jedem Binden bestimmte Optionen anzugeben. Wenn Sie diese Variable setzen, überprüft der Linker die Variable auf Optionen und erwartet, daß Optionen exakt so aufgeführt sind, wie Sie diese auf der Befehlszeile eintippen würden. In der LINK-Umgebungsvariablen können Sie keine Dateinamenargumente angeben.

Hinweis Eine Befehlszeilen-Option überschreibt die Auswirkung jeder Umgebungsvariablen-Option, mit der die Befehlszeilen-Option kollidiert. Die Befehlszeilen-Option /**SE**:256 zum Beispiel hebt die Auswirkung der Umgebungsvariablen-Option /**SE**:512 auf.

Um zu verhindern, daß eine Option der Umgebungsvariablen verwendet wird, müssen Sie die Umgebungsvariable selbst neu setzen.

Beispiel

Im folgenden Beispiel wird die Datei TEST.OBJ mit den Optionen **/SE**:256 und **/CO** gebunden. Anschließend wird die Datei PROG.OBJ sowohl mit der Option **/NOD** als auch mit den Optionen **/SE**:256 und **/CO** gebunden.

```
SET LINK=/SE:256 /CO
LINK TEST;
LINK /NOD PROG;
```

9.5.5.1 Wie Sie sich die Liste der Optionen ansehen können (/HE)

/HE[**LP**]

Die Option **/HE** weist den Linker an, eine Liste der verfügbaren Linker-Optionen auf dem Bildschirm anzuzeigen.

9.5.5.2 Während des Bindens anhalten (/PAU)

/PAU[**SE**]

Die Option **/PAU** weist den Linker an, den Bindevorgang vorübergehend zu unterbrechen und eine Meldung anzuzeigen, bevor er die ausführbare Datei auf die Diskette schreibt. Dies ermöglicht es Ihnen, eine neue Diskette, auf die die ausführbare Datei geschrieben wird, einzulegen.

Wenn Sie die Option **/PAUSE** angeben, gibt der Linker, bevor er die ausführbare Datei anlegt, die folgende Meldung aus:

```
.EXE-Datei wird erzeugt
Diskette in Laufwerk Buchstabe wechseln; <EINGABE> drücken
```

Der *Buchstabe* entspricht dem aktuellen Laufwerk. Der Linker setzt die Verarbeitung fort, wenn Sie die EINGABETASTE betätigen.

Hinweis Entfernen Sie auf keinen Fall die Diskette, auf der die List-Datei angelegt ist, oder die Diskette, die für die temporäre Datei verwendet wird.

Falls eine temporäre Datei auf der Diskette, die Sie tauschen möchten, angelegt ist, sollten Sie STRG+C betätigen, um den Bindevorgang zu beenden. Organisieren Sie Ihre Dateien so um, daß sowohl die temporäre Datei als auch die ausführbare Datei auf dieselbe Diskette geschrieben werden können. Versuchen Sie anschließend erneut zu binden.

9.5.5.3 Anzeigen von Informationen des Bindevorganges (/I)

/I[NFORMATION]

Die Option **/I** zeigt Informationen des Bindevorganges an, einschließlich der Phase des Bindens und der Namen der Objektdateien, die gebunden werden.

Diese Option hilft Ihnen, die Dateilage der Objektdateien, die gebunden werden, sowie die Reihenfolge, in der diese gebunden werden, zu bestimmen.

9.5.5.4 Unterdrücken der Linker-Anfragen (/B)

/B[ATCH]

Die Option **/B** teilt dem Linker mit, Sie nicht immer dann nach einem neuen Pfadnamen zu fragen, wenn der Linker eine von ihm benötigte Bibliothek oder Objektdatei nicht finden kann. Wenn diese Option verwendet wird, setzt der Linker die Ausführung einfach ohne die fragliche Datei fort.

Diese Option kann ungelöste externe Bezüge verursachen. Sie ist in erster Linie für Benutzer gedacht, die Stapel- oder MAKE-Dateien dazu einsetzen, viele ausführbare Dateien mit einem einzigen Befehl zu binden, und die nicht wünschen, daß der Linker die Verarbeitung unterbricht, falls er eine benötigte Datei nicht finden kann. Diese Option ist auch hilfreich, wenn Sie die **link**-Befehlszeile umleiten, um eine Datei mit Linker-Ausgaben für zukünftige Bezugnahme zu erzeugen. Diese Option verhindert jedoch nicht, daß der Linker Argumente nachfragt, die auf der **link**-Befehlszeile fehlen.

9.5.5.5 Quick-Bibliotheken anlegen (/Q)

/Q[UICKLIB]

Die Option **/Q** weist den Linker an, die von Ihnen angegebenen Objektdateien in einer Quick-Bibliothek zusammenzufassen. Wenn Sie die QuickBASIC-Umgebung starten, können Sie die Option **/l** auf der qb-Befehlszeile angeben, um die Quick-Bibliothek zu laden. Wenn Sie die Option **/Q** verwenden, dürfen Sie nicht vergessen, in der Liste der Bibliotheken BQLB40.LIB anzugeben, um die QuickBASIC-Unterstützungsroutinen für Quick-Bibliotheken einzubinden.

Weitere Informationen zum Anlegen und Laden von Quick-Bibliotheken finden Sie in Kapitel 8, "Quick-Bibliotheken".

Hinweis Sie können die Optionen **/EXEPACK** und **/Q** nicht zusammen verwenden.

9.5.5.6 Ausführbare Dateien packen (/E)

/E[XEPACK]

Die Option **/E** entfernt Folgen sich wiederholender Bytes (typischerweise Nullzeichen) und optimiert die "Ladezeit-Verschiebungstabelle", bevor die ausführbare Datei angelegt wird. Die Ladezeit-Verschiebungstabelle ist eine Tabelle mit Bezügen relativ zu dem Beginn des Programmes. Jeder Bezug wechselt, wenn das ausführbare Abbild in den Speicher geladen ist und eine aktuelle Adresse für den Eingangspunkt zugewiesen ist.

Mit dieser Option gebundene ausführbare Dateien sind kleiner und werden schneller geladen als Dateien, die ohne diese Option gebunden sind.

9.5.5.7 Segment-Packen ausschalten (/NOP)

/NOP[ACKCODE]

Die Option **/NOP** ist normalerweise nicht notwendig, da das Code-Segment-Packen normalerweise ausgeschaltet ist. Falls eine Umgebungsvariable, wie zum Beispiel LINK, Code-Segment-Packen automatisch einschaltet, können Sie die Option **/NOP** dazu einsetzen, das Segment-Packen wieder auszuschalten.

9.5.5.8 Die üblichen BASIC-Bibliotheken ignorieren (/NOD)

/NOD[EFAULTLIBRARYSEARCH]

Wenn BC eine Objektdatei anlegt, fügt es die Namen der "Standard"-Bibliotheken ein - Bibliotheken, die der Linker durchsucht, um externe Bezüge aufzulösen. Die Option **/NOD** weist den Linker an, *keine* in einer Objektdatei angegebene Bibliothek zu durchsuchen, um externe Bezüge aufzulösen.

Normalerweise arbeiten QuickBASIC-Programme ohne die Standard-Bibliotheken von QuickBASIC (BRUN40.LIB und BCOM40.LIB) nicht richtig. Daher sollten Sie ausdrücklich den Pfadnamen zu der erforderlichen Standard-Bibliothek angeben, wenn Sie die Option **/NOD** verwenden.

9.5.5.9 Die maximale Anzahl an Segmenten setzen (/SE)

/SE[GMENTS]:*Anzahl*

Die Option **/SE** steuert die Anzahl an Segmenten, die der Linker einem Programm zubilligt. Der Vorgabewert beträgt 128, Sie können *Anzahl* aber auf jeden beliebigen Wert (angegeben als Dezimal-, Oktal- oder Hexadezimalzahl) in dem Bereich von 1 bis 1024 (dezimal) setzen.

Für jedes Segment muß der Linker Speicherplatz zuweisen, um Segment-Informationen festzuhalten. Wenn Sie die Segmentgrenze höher als 128 setzen, weist der Linker mehr Platz für Segment-Informationen zu. Für Programme mit weniger als 128 Segmenten können Sie den Speicherbetrag, den der Linker benötigt, minimieren, indem Sie *Anzahl* so setzen, daß die aktuelle Anzahl an Programm-Segmenten wiedergegeben wird. Der Linker gibt eine Fehlermeldung aus, falls diese Anzahl zu hoch für den Speicherbetrag ist, der dem Linker zur Verfügung steht.

9.5.5.10 Eine Map-Datei erstellen (/M)

/M[**AP**][:*Anzahl*]

Die Option **/M** erstellt eine Map-Datei. Eine Map-Datei listet die Programm-Segmente in der Reihenfolge auf, in der sie in dem Lademodul erscheinen. Das optionale Feld *Anzahl* legt die maximale Anzahl globaler Symbole fest, die der Linker sortieren kann. Standardmäßig liegt diese Obergrenze bei 2048. Wenn die Anzahl an Symbolen diese Obergrenze überschreitet, dann erzeugt der Linker nur eine unsortierte Liste. Wenn Sie für *Anzahl* einen Wert angeben, dann enthält *Mapdatei* eine Liste von Symbolen, die nach Adressen sortiert sind (vorausgesetzt, daß *Anzahl* groß genug ist); die Map-Datei enthält keine nach Namen sortierte Liste. Es folgt ein Beispiel für eine Map-Datei:

```
Start  Stop   Length Name      Class
00000H 01E9FH 01EA0H _TEXT     CODE
01EA0H 01EA0H 00000H C_ETEXT   ENDCODE
.
.
.
.
```

Die Informationen der mit `Start` (Beginn) und `Stop` (Ende) bezeichneten Spalten zeigen die 20-Bit-Adresse (hexadezimal) für jedes Segment relativ zum Beginn des Lademoduls. Das Lademodul beginnt mit der Dateilage Null. Die mit `Length` bezeichnete Spalte gibt die Länge des Segmentes in Bytes an. Die mit `Name` bezeichnete Spalte gibt den Namen des Segmentes an, und die mit `Class` bezeichnete Spalte gibt Informationen über den Typ des Segmentes an. Weitere Informationen zu Gruppen, Segmenten und Klassen finden Sie im *Microsoft MS-DOS Programmer's Reference.*

Die Startadresse sowie der Name jeder Gruppe erscheinen unter der Liste der Segmente. Nachfolgend ein Beispiel-Listing für Gruppe:

```
Origin   Group
01EA:0   DGROUP
```

In diesem Beispiel ist DGROUP der Name der Datengruppe.

Die weiter unten gezeigte Map-Datei enthält zwei Listen globaler Symbole: Die erste Liste enthält in Reihenfolge der ASCII-Zeichen sortierte Symbolnamen; die zweite ist nach Symboladressen sortiert. Die Bezeichnung Abs erscheint neben den Namen absoluter Symbole (Symbole, die 16-Bit-Konstantenwerte enthalten, die nicht mit Programmadressen verknüpft sind).

Viele der globalen Symbole, die in der Map-Datei erscheinen, sind Symbole, die intern von dem Compiler und Linker verwendet werden. Diese Symbole beginnen normalerweise mit den Zeichen B$ oder enden mit QQ.

```
Address              Publics by Name
01EA:0096            STKHQQ
0000:1D86            B$Shell
01EA:04B0            _edata
01EA:0910            _end
.
.
.
01EA:00EC            __abrkp
01EA:009C            __abrktb
01EA:00EC            __abrktbe
0000:9876    Abs     __acrtmsg
0000:9876    Abs     __acrtused
.
.
.
01EA:0240            ___argc
01EA:0242            ___argv

Address              Publics by Value
0000:0010            _main
0000:0047            _htoi
.
.
.
```

Die Adressen der externen Symbole liegen in dem Format *Rahmen:Offset* vor und zeigen die Dateilage des Symbols relativ zu Null (der Beginn des Lademoduls) an.

Nach der Liste der Symbole gibt die Map-Datei den Eingangspunkt des Programmes an, wie in dem folgenden Beispiel gezeigt:

```
Program entry point at 0000:0129
```

(Eingangspunkt des Programmes bei 0000:0129)

Eine Map-Datei kann auch festgelegt werden, indem der Name einer Map-Datei auf der **link**-Befehlszeile angegeben wird, oder indem der Name einer Map-Datei als Antwort auf die Anfrage "List-Datei" eingegeben wird.

9.5.5.11 Einfügen von Zeilennummern in eine Map-Datei (/LI)

/LI[**NENUMBERS**]

Die Option **/LI** erzeugt eine Map-Datei und fügt die Zeilennummern sowie die zugeordneten Adressen des Quellprogrammes ein. Falls Sie in einzelnen Schritten kompilieren und binden, hat diese Option nur dann Auswirkungen, wenn Sie Objektdateien binden, die mit der Option **/M** kompiliert sind.

9.5.5.12 Angrenzende Segmente packen (/PAC)

/[**NO**]**PAC**[**KCODE**][**:***Anzahl*]

Die Option **/PAC** weist den Linker an, benachbarte Code-Segmente zu gruppieren. Code-Segmente in derselben Gruppe benutzen gemeinsam dieselbe Segmentadresse; alle Offsetadressen werden anschließend nach Bedarf nach oben ausgerichtet. Als Ergebnis benutzen viele Befehle, die andernfalls unterschiedliche Segmentadressen haben würden, dieselbe Segmentadresse gemeinsam.

Falls angegeben, ist *Anzahl* das Größenlimit für Gruppen, die von **/PAC** gebildet werden. Der Linker beendet das Hinzufügen von Segmenten an eine bestimmte Gruppe, sobald er kein Segment zu der Gruppe hinzufügen kann, ohne *Anzahl* zu überschreiten. An diesem Punkt beginnt der Linker, mit den verbleibenden Code-Segmenten eine neue Gruppe zu bilden. Wenn *Anzahl* nicht angegeben ist, ist der Vorgabewert 65.530.

Obwohl der Linker benachbarte Segmente nicht packt, solange Sie dies nicht ausdrücklich angeben, können Sie die Option **/NOPACKCODE** dazu verwenden, das Segment-Packen auszuschalten, wenn Sie zum Beispiel in der Umgebungsvariablen LINK die Option **/PAC** angegeben haben.

9.5.5.13 Den CodeView-Debugger einsetzen (/CO)

/CO[**DEVIEW**]

Die Option **/CO** bereitet eine ausführbare Datei für das Debuggen mit dem CodeView-Debugger vor. Falls Sie in separaten Schritten kompilieren und binden, hat diese Option nur dann Auswirkungen, wenn Sie Objektdateien binden, die mit der **/zi**-Option des Befehls **bc** kompiliert sind. Auch sollte diese Option nicht zusammen mit der **/Q**-Option des **link**-Befehls verwendet werden, da eine Quick-Bibliothek nicht mit dem CodeView-Debugger auf Fehler untersucht werden kann.

9.5.5.14 Schreibweise unterscheiden (/NOI)

/NOI[**GNORECASE**]

Die Option **/NOI** weist den Linker an, zwischen Groß- und Kleinbuchstaben zu unterscheiden; zum Beispiel würde der Linker die Namen ABC, abc und Abc als drei unterschiedliche Namen betrachten. Wenn Sie binden, geben Sie die Option **/NOI** *nicht* auf der **link**-Befehlszeile an.

9.5.6 Andere Optionen der link-Befehlszeile

Nicht alle Optionen des Befehls **link** sind für die Verwendung mit QuickBASIC geeignet. Die folgenden Linker-Optionen können mit Microsoft-QuickBASIC-Programmen verwendet werden; sie sind jedoch niemals erforderlich, da sie Aktionen veranlassen, die der **bc**-Befehl bzw. QuickBASIC automatisch durchführen:

/CP[**ARMAXALLOC**]

Setzt die maximale Anzahl an 16-Byte-Paragraphen, die das Programm benötigt, wenn es an *Anzahl*, eine Ganzzahl zwischen 1 und einschließlich 65.535, in den Speicher geladen wird. Das Betriebssystem verwendet diesen Wert, wenn es Speicherplatz für dieses Programm zuweist, bevor das Programm geladen wird. Obwohl Sie diese Option auf der **link**-Befehlszeile verwenden können, hat die Option keine Auswirkungen, weil Ihr BASIC-Programm den Speicher kontrolliert, während es läuft.

/DO[**SSEG**]

Veranlaßt, daß Segmente mit den Vorgabewerte für Microsoft-Hochsprachen-Produkte angeordnet werden. QuickBASIC-Programme verwenden diese Segment-Anordnung immer standardmäßig.

/ST[**ACK**]:*Anzahl*

Legt für Ihr Programm die Größe des Stapels fest, wobei *Anzahl* ein beliebiger positiver Wert (dezimal, oktal oder hexadezimal) bis zu 65.535 (dezimal) ist, der die Größe des Stapels in Bytes darstellt. Die BASIC-Standard-Bibliothek setzt die Standardgröße des Stapels auf 2K.

Verwenden Sie die folgenden Linker-Optionen nicht, wenn Sie mit BC kompilierte Objektdateien binden. Diese Optionen sind nur für Objektdateien geeignet, die mit dem Microsoft Macro Assembler (MASM) erstellt wurden.

/DS[**ALLOCATE**]

Lädt alle Daten, beginnend am oberen Ende des Standard-Datensegmentes.

HI[GH]

Plaziert die ausführbare Datei so hoch wie möglich im Speicher.

/NOG[ROUPASSOCIATION]

Weist den Linker an, Gruppenverbände zu ignorieren, wenn Daten- und Codegrößen Adressen zugewiesen werden.

/O[VERLAYINTERRUPT]:*Anzahl*

Legt für die Übergabe der Kontrolle an ein Overlay eine andere Interrupt-Nummer als 0x3F fest.

9.6 Selbständige Bibliotheken verwalten: LIB

Der Microsoft-Bibliotheksmanager (LIB) verwaltet den Inhalt einer selbständigen Bibliothek.

Eine selbständige Bibliothek besteht aus "Objektmodulen" - das heißt, Objektdateien, die zusammengefaßt sind, um eine Bibliothek zu bilden. Anders als eine Objektdatei existiert ein Objektmodul nicht unabhängig von der Bibliothek, zu der das Objektmodul gehört, und es hat keinen Pfadnamen bzw. keine Erweiterung, die mit seinem Dateinamen verknüpft ist. Mit LIB können Sie

- Objektdateien kombinieren, um eine neue Bibliothek zu erstellen.
- Objektdateien in eine existierende Bibliothek einfügen.
- Objektmodule einer existierenden Bibliothek löschen oder ersetzen.
- Objektmodule aus einer existierenden Bibliothek herausnehmen und diese in separaten Objektdateien ablegen.
- Inhalte zweier existierender Bibliotheken in einer neuen Bibliothek zusammenfassen.

Wenn Sie eine existierende Bibliothek aktualisieren, führt LIB alle seine Operationen mit einer Kopie der Bibliothek durch. Diese Vorgehensweise stellt sicher, daß Sie eine Sicherungskopie jeder Bibliothek haben, die Sie aktualisieren, falls mit der aktualisierten Version der Bibliothek Probleme auftreten.

9.6.1 LIB starten

Sie können die Eingabe für den Befehl **LIB** auf eine der folgenden Arten vornehmen:

- Geben Sie die Eingaben in einer Befehlszeile folgender Form an:

  LIB *alteBibl* [/**P**[**AGESIZE**]:*Nummer*] [*Befehle*][,[*List-Datei*][,[*neueBibl*]]][;]

  Die Befehlszeile darf nicht länger als 128 Zeichen sein.

- Geben Sie `lib` ein und antworten auf die folgenden Anfragen:

  ```
  Bibliotheksname:
  Operationen:
  List-Datei:
  Ausgabebibliothek:
  ```

  Um für eine Anfrage mehrere Dateien anzugeben, schreiben Sie ein kaufmännisches Und-Zeichen (&) an das Ende der Zeile. Die Anfrage erscheint erneut auf der nächsten Zeile, und Sie können mit der Eingabe für die Anfrage fortfahren.

- Installieren Sie eine Datei mit Antworten auf die Anfragen des **LIB**-Befehls, die als "Antwortdatei" (Response File) bezeichnet wird, und geben Sie anschließend einen **LIB**-Befehl der folgenden Form ein:

  LIB @ *Dateiname*

  Hierbei ist *Dateiname* der Name der Antwortdatei. Die Antworten müssen in derselben Reihenfolge sein wie die oben erläuterten LIB-Anfragen. Außerdem können Sie den Namen einer Antwortdatei nach jeder Linker-Anfrage oder an beliebiger Stelle auf der **LIB**-Befehlszeile eingeben.

Die folgende Tabelle zeigt die Eingaben, die Sie auf der **LIB**-Befehlszeile oder als Antwort auf jede Anfrage vornehmen müssen.

Feld	***Anfrage***	***Eingabe***
alteBibl	"Bibliotheksname"	Name der Bibliothek, die Sie verändern oder anlegen. Wenn diese Bibliothek nicht existiert, fragt LIB, ob Sie diese anlegen möchten. Geben Sie den Buchstaben "y (j)" ein, um eine neue Bibliothek anzulegen, oder den Buchstaben "n", um LIB zu beenden. Diese Meldung wird unterdrückt, wenn Sie Befehlszeichen, ein Komma oder ein Semikolon hinter dem Namen der Bibliothek eingeben. Ein Semikolon weist LIB an, für die Bibliothek eine Prüfung auf Übereinstimmung vorzunehmen; in diesem Fall gibt LIB eine Meldung aus, wenn es Fehler in irgendeinem Bibliotheksmodul findet.
/P:*Zahl*	**/P:***Zahl* hinter der Anfrage "Bibliotheksname"	Die Seitengröße der Bibliothek. Dies setzt die Seitengröße der Bibliothek auf *Zahl*, wobei *Zahl* eine ganzzahlige Potenz von 2 zwischen 16 und 32.768 einschließlich sein kann. Die Standard-Seitengröße für eine neue Bibliothek beträgt 16 Bytes. Module in dieser Bibliothek sind immer so ausgerichtet, daß sie auf einer Position beginnen, die ein Vielfaches der Seitengröße (in Bytes) ab Dateibeginn darstellt.
Befehle	"Operationen"	Befehlssymbole sowie Objektdateien, die LIB mitteilen, welche Änderungen in der Bibliothek durchgeführt werden sollen.
List-Datei	"List-Datei"	Name einer List-Datei für Querverweise. Es wird keine List-Datei erzeugt, wenn Sie keinen Dateinamen angeben.
neuBibl	"Ausgabebibliothek"	Name der veränderten Bibliothek, die LIB als Ausgabe erzeugt. Falls Sie keinen neuen Dateinamen angeben, wird die ursprüngliche, unveränderte Datei in einer Bibliotheksdatei mit demselben Namen, aber der Erweiterung .BAK, die die Erweiterung .LIB ersetzt, gespeichert.

Wenn Sie eine Antwortdatei verwenden, muß jede Antwort den in der obigen Tabelle beschriebenen Regeln folgen.

9.6.2 Übliche Vorgaben für LIB

LIB hat eigene eingebaute Antworten, manchmal als Vorgaben bezeichnet. Sie können diese Vorgaben für jede von LIB benötigte Information auf eine der folgenden Arten wählen:

- Um für einen Eintrag der Befehlszeile die Vorgabe zu wählen, lassen Sie den Dateinamen oder die Dateinamen vor dem Eintrag aus und geben nur das erforderliche Komma ein. Die einzige Ausnahme zu dieser Regel ist der Vorgabe für den Eintrag der *List-Datei*: Wenn Sie ein Komma als Platzhalter für diesen Eintrag verwenden, erzeugt LIB eine List-Datei mit Querverweisen.
- Um die Vorgabe für jede Anfrage zu wählen, betätigen Sie einfach die EINGABETASTE.
- Um die Vorgabewerte für alle Einträge oder Anfragen der Befehlszeile, die einem Eintrag oder einer Anfrage folgen, zu wählen, geben Sie ein Semikolon (;) nach dem Eintrag oder der Anfrage ein. Das Semikolon sollte das letzte Zeichen auf der Befehlszeile sein.

Die folgende Liste zeigt die Vorgabewerte, die LIB für List-Dateien mit Querverweisen und Ausgabebibliotheken verwendet.

Datei	***Vorgabe***
Liste mit Querverweisen	Der spezielle Dateiname NUL, der dem Linker mitteilt, *keine* List-Datei mit Querverweisen anzulegen.
Ausgabebibliothek	Der Eintrag *alteBibl* oder die Antwort auf die Anfrage " Bibliotheksname".

9.6.3 Befehlssymbole

Um LIB mitzuteilen, welche Änderungen Sie in der Bibliothek durchführen möchten, geben Sie ein Befehlssymbol, wie zum Beispiel +, -, -+, * oder -*, direkt gefolgt von einem Modulnamen, Namen einer Objektdatei oder Namen einer Bibliothek ein. Sie können mehr als eine Operation in beliebiger Reihenfolge angeben.

Die folgende Liste zeigt jedes **LIB**-Befehlssymbol, den Typ des mit dem Symbol anzugebenden Dateinamens, und was das Symbol bewirkt:

Befehl	***Bedeutung***
+{*Objdatei* | *Bibl*}	Fügt die angegebene Objektdatei zu der eingegebenen Bibliothek hinzu und läßt diese Objektdatei das letzte Modul in der Bibliothek werden, falls die Objektdatei mit dem Namen einer Objektdatei angegeben ist. Sie können einen Pfadnamen für den Namen der Objektdatei verwenden. Da LIB automatisch die Erweiterung .OBJ anfügt, können Sie die Erweiterung für den Namen der Objektdatei auslassen. Falls mit einem Bibliotheksnamen angegeben, fügt das Pluszeichen (+) den Inhalt dieser Bibliothek in die eingegebene Bibliothek ein. Der Name der Bibliothek muß die Erweiterung .LIB haben.
-*Modul*	Löscht das angegebene Modul aus der eingegebenen Bibliothek. Ein Modulname hat keinen Pfadnamen bzw. keine Erweiterung.
-+*Modul*	Ersetzt das angegebene Modul in der eingegebenen Bibliothek. Modulnamen haben keinen Pfadnamen bzw. keine Erweiterungen. LIB löscht das angegebene Modul und fügt anschließend die Objektdatei an, die denselben Namen wie das Modul hat. Von der Objektdatei wird angenommen, das sie die Erweiterung .OBJ hat und sich in dem aktuellen Arbeitsverzeichnis befindet.
**Modul*	Kopiert das angegebene Modul aus der Bibliothek in eine Objektdatei, die sich im aktuellen Arbeitsverzeichnis befindet. Das Modul verbleibt in der Bibliotheksdatei. Wenn LIB das Modul in eine Objektdatei kopiert, fügt es die Erweiterung .OBJ an. Sie können weder die Erweiterung .OBJ noch das Ziellaufwerk noch den Pfadnamen, der für die Objektdatei gegeben ist, überschreiben. Später jedoch können Sie die Datei umbenennen oder an jede beliebige von Ihnen gewünschte Stelle kopieren.
-**Modul*	Bewegt das angegebene Objektmodul aus der Bibliothek in eine Objektdatei. Diese Operation ist identisch mit dem Kopieren des Moduls in eine Objektdatei – wie oben beschrieben – und anschließendem Löschen des Moduls aus der Bibliothek.

Beispiele

Das folgende Beispiel verwendet das Befehlssymbol zum Ersetzen (-+), um LIB anzuweisen, das Modul HEAP in der Bibliothek LANG.LIB zu ersetzen. Das Programm LIB löscht das Modul HEAP aus der Bibliothek und fügt anschließend die Objektdatei HEAP.OBJ als neues Modul in die Bibliothek ein. Das Semikolon am Ende der Befehlszeile zeigt LIB an, daß es die Vorgaben für die verbleibenden Anfragen verwenden soll. Das bedeutet, daß keine List-Datei erzeugt wird, und daß die Änderungen in die ursprüngliche Bibliotheksdatei geschrieben werden, anstatt eine neue Bibliothek zu erzeugen.

```
LIB LANG-+HEAP;
```

Die folgenden Beispiele führen dieselbe Funktion wie das erste Beispiel in diesem Abschnitt aus, allerdings in zwei separaten Operationen, die die Befehlssymbole Hinzufügen (+) und Löschen (-) verwenden. Die Ergebnisse dieser Beispiele sind identisch, da Operationen zum Löschen immer vor Operationen zum Hinzufügen ausgeführt werden, unabhängig von der Reihenfolge dieser Operationen auf der Befehlszeile. Diese Reihenfolge der Ausführung verhindert Konflikte, wenn eine neue Version eines Modules eine alte Version in der Bibliotheksdatei ersetzt.

```
LIB LANG-HEAP+HEAP;
LIB LANG+HEAP-HEAP;
```

Das folgende Beispiel veranlaßt LIB, eine Prüfung auf Übereinstimmung mit der Bibliotheksdatei FOR.LIB vorzunehmen. Es wird keine andere Aktion durchgeführt. LIB zeigt jeden Übereinstimmungsfehler an, den es findet, und kehrt auf Betriebssystemebene zurück.

```
LIB FOR;
```

Das folgende Beispiel teilt LIB mit, eine Prüfung auf Übereinstimmung mit der Bibliotheksdatei LANG.LIB durchzuführen und anschließend eine List-Datei mit Querverweisen zu erzeugen, die LQUER.PUB heißt.

```
LIB LANG,LQUER.PUB
```

Das nächste Beispiel weist LIB an, das Modul PUMPEN aus der Bibliothek ERSTE.LIB in eine PUMPEN.OBJ benannte Objektdatei zu bewegen. Das Modul PUMPEN wird bei diesem Vorgang aus der Bibliothek entfernt. Das Modul MEHR wird aus der Bibliothek in eine Objektdatei kopiert, die MEHR.OBJ heißt; das Modul verbleibt in der Bibliothek. Die geänderte Bibliothek heißt ZWEITE.LIB. Sie enthält alle Module der Bibliothek ERSTE.LIB, mit Ausnahme des Moduls PUMPEN, das entfernt wurde, indem das Befehlssymbol Bewegen (-*) verwendet wurde. Die ursprüngliche Bibliothek ERSTE.LIB bleibt unverändert.

```
LIB ERSTE -*PUMPEN *MEHR, ,ZWEITE
```

Der Inhalt der folgenden Antwortdatei veranlaßt LIB, das Modul HEAP aus der Bibliotheksdatei LIBFOR.LIB zu löschen, das Modul SCHWACH herauszunehmen, und es in einer SCHWACH.OBJ benannten Objektdatei abzulegen, sowie die Objektdateien CURSOR.OBJ und HEAP.OBJ als die letzten beiden Module an die Bibliothek anzuhängen. Zum Schluß erzeugt LIB eine List-Datei mit Querverweisen, die QUERLST heißt.

```
LIBFOR
+CURSOR+HEAP-HEAP*SCHWACH
QUERLST
```

9.6.4 List-Dateien mit Querverweisen

Eine List-Datei mit Querverweisen hilft Ihnen nachzuverfolgen, welche Routinen sich in einer selbständigen Bibliothek befinden und aus welchen ursprünglichen Objektdateien diese stammen. Eine List-Datei mit Querverweisen enthält die folgenden Listen:

- Eine alphabetische Liste aller globalen Symbole in der Bibliothek. Jedem Symbolnamen folgt der Name des Moduls, in dem auf das Symbol Bezug genommen wird.
- Eine Liste der Module in der Bibliothek. Unter jedem Modulnamen befindet sich eine alphabetische Liste der in diesem Modul definierten globalen Symbole.

9.6.5 Die Größe der Bibliotheksseite setzen

Die Seitengröße einer Bibliothek hat Auswirkungen auf die Ausrichtung von Modulen, die in der Bibliothek gespeichert sind. Module in der Bibliothek sind immer so ausgerichtet, daß sie auf einer Position beginnen, die ein Vielfaches der Seitengröße (in Bytes) ab Dateibeginn darstellt. Die vorgegebene Seitengröße für eine neu angelegte Bibliothek beträgt 16 Bytes.

Sie können eine andere Bibliotheks-Seitengröße setzen, während Sie eine Bibliothek anlegen oder die Seitengröße einer existierenden Bibliothek ändern, indem Sie die folgende Option nach dem *altBibl*-Eintrag auf der **LIB**-Befehlszeile oder nach dem Namen, den Sie als Antwort auf die Anfrage "Bibliotheksname" eingeben, anfügen:

/P[AGESIZE]:*Zahl*

Zahl legt die neue Bibliotheks-Seitengröße fest. Sie muß ein ganzzahliger Wert sein, der eine Potenz von 2 zwischen den Werten 16 und 32.768 darstellt.

Die Bibliotheks-Seitengröße bestimmt die Anzahl an Modulen, die die Bibliothek enthalten kann; daher ermöglicht die Vergrößerung der Seitengröße es Ihnen, mehr Module in die Bibliothek einzufügen. Je größer die Seitengröße, desto größer ist jedoch der ungenutzte Speicherplatz in der Bibliothek (im Durchschnitt *Seitengröße*/2 Bytes). In den meisten Fällen sollten Sie eine kleine Seitengröße verwenden, es sei denn, Sie müssen eine sehr große Anzahl an Modulen in einer Bibliothek ablegen.

Außerdem bestimmt die Seitengröße die maximal mögliche Größe der Bibliothek. Diese Obergrenze ist *Zahl* * 65.536. Wenn Sie LIB zum Beispiel mit der Option **/P**:16 aufrufen, muß die Bibliothek kleiner als ein Megabyte (16 * 65.536 Bytes) sein.



Teil 4: Anhänge



Anhang A: Übertragen von BASICA-Programmen nach QuickBASIC

Die Sprache BASIC in Form von QuickBASIC ist generell kompatibel mit den BASIC-Interpretern, die auf IBM Personal Computern sowie Kompatiblen eingesetzt werden: IBM Advanced Personal Computer BASIC Version A3.00 (allgemein bekannt als BASICA) und Microsoft GW-BASIC. Einige Änderungen sind jedoch aufgrund interner Unterschiede zwischen QuickBASIC und diesen BASIC-Interpretern notwendig. Da die Änderungen für beide Interpreter gleich vorgenommen werden müssen, wird in diesem Anhang der Begriff BASICA verwendet, der sich sowohl auf GW-BASIC als auch auf BASICA bezieht.

Dieser Anhang bietet die folgenden Informationen zur Übertragung von Dateien:

- Kompatibles Format einer Quelldatei
- Anweisungen und Funktionen, die nicht zulässig sind bzw. Änderungen erfordern.

Die folgenden Abschnitte beschreiben nur die erforderlichen Änderungen, um ein BASICA-Programm mit QuickBASIC Version 4.0 zu kompilieren und zu starten. Informationen darüber, wie QuickBASIC-Eigenschaften verwendet werden, um Ihre existierenden BASICA-Programme zu verbessern, finden Sie in *Programmieren in BASIC: Ausgewählte Themen.*

A.1 Format einer Quelldatei

QuickBASIC Version 4.0 erwartet, daß die Quelldatei im ASCII-Format oder im QuickBASIC-Format vorliegt. Wenn Sie eine Datei mit BASICA erstellen, muß diese mit der Option ‚A abgespeichert werden; andernfalls komprimiert BASICA den Text Ihres Programmes in einem besonderen Format, das QuickBASIC nicht lesen kann. Falls dies passiert, sollten Sie BASICA erneut laden und die Datei diesmal mit der Option ‚A im ASCII-Format abspeichern. Zum Beispiel speichert der folgende BASICA-Befehl die Datei MEINPROG.BAS im ASCII-Format ab:

```
SAVE "MEINPROG.BAS",A
```

A.2 Kassetten-BASIC

QuickBASIC unterstützt keine Kassetten-Geräte.

A.3 Anweisungen und Funktionen, die in QuickBASIC nicht zulässig sind

Die unten aufgeführten Anweisungen und Funktionen dürfen nicht in einem QuickBASIC-Programm verwendet werden, da sie Editierungen in der Quelldatei durchführen, in die Programmausführung eingreifen, sich auf Kassetten-Geräte beziehen oder mit einer in der QuickBASIC-Umgebung bereits bereitgestellten Hilfe übereinstimmen.

AUTO	**LIST**	**NEW**
CONT	**LLIST**	**RENUM**
DEF USR	**LOAD**	**SAVE**
DELETE	**MERGE**	**USR**
EDIT	**MOTOR**	

A.4 Anweisungen, die Änderungen erfordern

Falls Ihr BASICA-Programm irgendeine der in der folgenden Tabelle aufgeführten Anweisungen enthält, müssen Sie Ihren Quellcode wahrscheinlich ändern, bevor Sie das Programm kompilieren und starten können. Eine genaue Beschreibung dieser Anweisungen finden Sie im *BASIC-Befehlsverzeichnis.*

Anweisungen	*Änderungen*
BLOAD/ BSAVE	Dateilagen im Speicher können in QuickBASIC anders sein.
CALL *Name*	Das Argument *Name* ist der Name der **FUNCTION**- oder **SUB**-Prozedur, die aufgerufen wird.
CHAIN	QuickBASIC unterstützt nicht die Optionen **ALL, MERGE, DELETE** oder *Zeilennummer*.
COMMON	**COMMON**-Anweisungen müssen vor der ersten ausführbaren Anweisung erscheinen.
DEF*Typ*	**DEF***Typ*-Anweisungen sollten an den Anfang der BASICA-Quelldatei geschrieben werden.

Anweisungen	*Änderungen*
DIM	Alle **DIM**-Anweisungen, die statische Datenfelder deklarieren, müssen am Beginn des QuickBASIC-Programms erscheinen. Informationen zu **$DYNAMIC** finden Sie in Kapitel 2, "Datentypen", im *BASIC-Befehlsverzeichnis*.
DRAW	QuickBASIC verlangt, daß die Funktion **VARPTR$** für eingebettete Variablen verwendet wird.
PLAY	QuickBASIC verlangt, daß die Funktion **VARPTR$** für eingebettete Variablen verwendet wird.
RESUME	Falls in einer einzeiligen Funktion ein Fehler auftritt, versucht QuickBASIC, die Programmausführung mit der Zeile fortzusetzen, die die Funktion enthält.
RUN	Für ausführbare Dateien, die mit QuickBASIC erzeugt wurden, darf das Objekt einer **RUN**- oder **CHAIN**-Anweisung keine .BAS-Datei sein; es muß eine ausführbare Datei sein. Die BASICA-Option R wird nicht unterstützt. Während es in der Umgebung ausgeführt wird, ist das Objekt einer **RUN**- oder **CHAIN**-Anweisung immer noch eine .BAS-Datei. **RUN** {*Zeilennummer* | *Zeilenmarke*} wird jedoch von QuickBASIC unterstützt; diese Anweisung startet das Programm neu ab der angegebenen Zeile.

Anhang B: Unterschiede zu früheren QuickBASIC-Versionen

QuickBASIC Version 4.0 enthält viele neue Eigenschaften und Verbesserungen gegenüber vorhergehenden QuickBASIC-Versionen. Dieser Anhang beschreibt die Unterschiede zwischen den QuickBASIC-Versionen 2.0 und 4.0. Solange nicht anders erwähnt, beziehen sich Unterschiede zwischen den Versionen 2.0 und 4.0 auch auf Unterschiede zwischen den Versionen 3.0 und 4.0.

Dieser Anhang bietet die folgenden Informationen zu Verbesserungen von QuickBASIC Version 4.0:

- Produktfähigkeiten und -eigenschaften
- Erweiterungen der Umgebung
- Verbesserungen beim Kompilieren und Debuggen
- Sprachänderungen
- Datei-Kompatibilität

B.1 Neue Eigenschaften

Dieser Abschnitt beschreibt alle neuen Eigenschaften der Microsoft QuickBASIC-Version 4.0. Die Eigenschaften sind in der folgenden Tabelle aufgeführt und werden in den Abschnitten B.1.1 bis B.1.17 beschrieben.

Eigenschaft	*QuickBASIC Version 2.0*	*QuickBASIC Version 3.0*	*QuickBASIC Version 4.0*
Benutzerdefinierte Typen	Nein	Nein	Ja
IEEE-Format, Unterstützung des mathematischen Koprozessors	Nein	Ja	Ja
Direkte Hilfe	Nein	Nein	Ja
Lange (32-Bit) Ganzzahlen	Nein	Nein	Ja
Zeichenketten fester Länge	Nein	Nein	Ja
Syntaxprüfung bei der Eingabe	Nein	Nein	Ja
Binär-Datei-E/A	Nein	Nein	Ja
FUNCTION-Prozeduren	Nein	Nein	Ja
CodeView-Unterstützung	Nein	Nein	Ja
Kompatibilität zu anderen Sprachen	Nein	Nein	Ja

Eigenschaft	*QuickBASIC Version 2.0*	*QuickBASIC Version 3.0*	*QuickBASIC Version 4.0*
Mehrere Module im Speicher	Nein	Nein	Ja
Kompatibilität mit ProKey, SideKick und SuperKey	Nein	Ja	Ja
Einfüge-/Überschreibemodus	Nein	Ja	Ja
Tastaturschnittstelle im WordStar-Stil	Nein	Nein	Ja
Rekursion	Nein	Nein	Ja
Fehler-Listings während separater Kompilierung	Nein	Ja	Ja
Assembler-Listings während separater Kompilierung	Nein	Ja	Ja

B.1.1 Benutzerdefinierte Typen

Die Anweisung **TYPE** ermöglicht es Ihnen, zusammengesetzte Datentypen zu erzeugen, indem Zeichenkettenelemente mit numerischen Elementen kombiniert werden. Solche Datentypen sind den Strukturen in der Sprache C ähnlich. Benutzerdefinierte Typen werden in Kapitel 2, "Datentypen", im *BASIC-Befehlsverzeichnis*, sowie Kapitel 3, "Datei- und Geräte-E/A", in *Programmieren in BASIC: Ausgewählte Themen* erläutert.

B.1.2 IEEE-Format und Unterstützung des mathematischen Koprozessors

Microsoft QuickBASIC bietet nun Zahlen im IEEE-Format sowie Unterstützung eines mathematischen Koprozessors an. Wenn kein Koprozessor vorhanden ist, emuliert QuickBASIC den Koprozessor.

Berechnungen, die von Programmen ausgeführt werden, die mit QuickBASIC Version 4.0 kompiliert sind, sind im allgemeinen genauer und können möglicherweise andere Ergebnisse hervorbringen, als Programme, die unter BASICA oder früheren QuickBASIC-Versionen laufen. Zahlen einfacher Genauigkeit im IEEE-Format bieten eine zusätzliche Dezimalziffer an Genauigkeit. Im Vergleich mit Zahlen doppelter Genauigkeit im Microsoft-Binär-Format bieten Zahlen doppelter Genauigkeit im IEEE-Format ein oder zwei zusätzliche Ziffern für die Mantisse und vergrößern den Bereich des Exponenten.

Es gibt zwei Möglichkeiten, QuickBASIC Version 4.0 mit Ihren alten Daten und Programmen zu verwenden:

1. Verwenden Sie die Option **/mbf**. Auf diese Art und Weise können Sie Ihre alten Programme und Datendateien verwenden, ohne Ihre Programme umzuschreiben oder Ihre Dateien zu ändern. Weitere Informationen finden Sie in Abschnitt B.1.2.3.
2. Verändern Sie Ihre Datendateien und verwenden Sie QuickBASIC Version 4.0, um Ihre Programme neu zu kompilieren. Auf lange Sicht stellt dies Kompatibilität mit zukünftigen QuickBASIC-Versionen sicher und kann schnellere Programme erzeugen. Es müssen nur Direktzugriffsdateien geändert werden, die reelle Zahlen im Binär-Format enthalten. Dateien, die nur ganze Zahlen oder Zeichenkettendaten enthalten, können ohne Änderungen verwendet werden. Weitere Informationen zur Konvertierung von Dateien und Durchführung von Dateieingabe und -ausgabe von reellen Zahlen finden Sie in Abschnitt B.1.2.4.

Hinweis Falls aus Ihrem Programm heraus Assembler-Prozeduren aufgerufen werden, die reelle Zahlen verwenden, müssen die Prozeduren so geschrieben sein, daß sie Zahlen im IEEE-Format einsetzen. Dies ist der Standard für den Microsoft Macro Assembler (MASM) Version 5.0. Mit früheren Versionen müssen Sie die Befehlszeilen-Option **/R** oder die Assembler-Direktive 8087 verwenden.

B.1.2.1 Bereiche der Zahlen im IEEE-Format

Zahlen im IEEE-Format besitzen einen größeren Bereich als Zahlen im Microsoft-Binär-Format, wie in der folgenden Liste gezeigt:

Zahlentyp	*Wertebereich*
Einfache Genauigkeit	$-3{,}37 * 10^{38}$ bis $-8{,}43 * 10^{-37}$
	Wirklich Null
	$8{,}43 * 10^{-37}$ bis $3{,}37 * 10^{38}$
Doppelte Genauigkeit	$-1{,}67 * 10^{308}$ bis $-4{,}19 * 10^{-307}$
	Wirklich Null
	$4{,}19 * 10^{-307}$ bis $1{,}67 * 10^{308}$

Werte einfacher Genauigkeit sind auf nahezu sieben Ziffern genau. Werte doppelter Genauigkeit sind genau auf entweder 15 oder 16 Ziffern.

Beachten Sie, daß Werte doppelter Genauigkeit im Exponenten drei Ziffern haben können. Dies könnte in **PRINT USING**-Anweisungen Probleme verursachen. Weitere Informationen finden Sie in Abschnitt B.1.2.2.

B.1.2.2 PRINT USING und Zahlen im IEEE-Format

Da Zahlen doppelter Genauigkeit im IEEE-Format größere Exponenten haben können als Zahlen doppelter Genauigkeit im Microsoft-Binär-Format, müssen Sie in **PRINT USING**-Anweisungen vielleicht ein besonderes Exponentialformat einsetzen. Verwenden Sie das neue Format, falls Ihre Programme Werte mit drei Ziffern im Exponenten ausdrucken. Um Zahlen mit drei Ziffern im Exponenten auszudrucken, müssen Sie fünf Einfügezeichen (^^^^^) anstelle von vier Einfügezeichen verwenden, um das Exponentialformat zu kennzeichnen:

```
PRINT USING "+#.######^^^^^", Umfang#
```

Falls ein Exponent zu groß für ein Feld ist, ersetzt QuickBASIC die erste Ziffer mit einem Prozentzeichen (%), um das Problem zu kennzeichnen.

B.1.2.3 Neukompilieren alter Programme mit /mbf

Alte Programme und Dateien arbeiten mit QuickBASIC Version 4.0 ohne Änderung, wenn Sie die Programme mit der Befehlszeilen-Option **/mbf** neu kompilieren.

Um zum Beispiel das als umwandlg.bas benannte Programm zu kompilieren, geben Sie die folgende Befehlszeile ein:

```
bc umwandlg /mbf;
```

Binden Sie das Programm anschließend wie gewohnt.

Um mit dem Dialogfeld **EXE-Datei erstellen** neu zu kompilieren, müssen Sie auf der **qb**-Befehlszeile die Option **/mbf** verwenden, wenn Sie QuickBASIC starten. Anschließend können Sie wie gewohnt kompilieren.

Die Option **/mbf** konvertiert Zahlen im Microsoft-Binär-Format in das IEEE-Format, während die Zahlen aus einer Direktzugriffsdatei gelesen werden, und konvertiert die Zahlen zurück in das Microsoft-Binär-Format, bevor sie in eine Datei geschrieben werden. Dies ermöglicht es Ihnen, Berechnungen mit Zahlen im IEEE-Format durchzuführen, während die Zahlen in Ihren Dateien im Microsoft-Binär-Format bleiben.

B.1.2.4 Dateien und Programme konvertieren

Wenn Sie sich dazu entschließen, Ihre Programme und Datendateien zu konvertieren und nicht die Befehlszeilen-Option **/mbf** einzusetzen, müssen Sie zwei Dinge tun:

1. Ihre Programme neu kompilieren.
2. Ihre Datendateien konvertieren.

Ihre alten QuickBASIC-Programme sollten ohne Änderungen kompiliert werden können. Verwenden Sie jedoch nicht die Option **/mbf**, wenn Sie neu kompilieren. Ihr Programm wird nicht mit Ihren neuen Datendateien arbeiten, falls Sie die Option **/mbf** verwenden.

Datendateien, die reelle Zahlen enthalten, müssen so konvertiert werden, daß reelle Zahlen im IEEE-Format gespeichert werden. QuickBASIC Version 4.0 enthält acht Funktionen, die Ihnen dabei helfen.

Hinweis Sie müssen Ihre Datendateien nicht konvertieren, wenn diese nur ganze Zahlen und Zeichenkettendaten enthalten. Nur Dateien, die reelle Zahlen enthalten, müssen konvertiert werden.

Version 4.0 stellt für das Lesen bzw. Schreiben reeller Zahlen aus bzw. auf Direktzugriffsdateien die bekannten Funktionen **CVS**, **CVD**, **MKS$** und **MKD$** zur Verfügung. In Version 4.0 behandeln diese Funktionen reelle Zahlen, die in Ihren Dateien gespeichert sind, als Zahlen im IEEE-Format und nicht als Zahlen im Microsoft-Binär-Format. Um Zahlen im Microsoft-Binär-Format zu behandeln, stellt Version 4.0 die Funktionen **CVSMBF**, **CVDMBF**, **MKSMBF$** und **MKDMBF$** zur Verfügung.

Mit diesen Funktionen können Sie ein kurzes Programm schreiben, das Sätze aus der alten Datei liest (und, falls notwendig, das Microsoft-Binär-Format verwendet), die reellen Zahlen in das IEEE-Format konvertiert, diese Zahlen in einem neuen Satz ablegt und den neuen Satz herausschreibt.

Beispiele

Das folgende Programm kopiert eine alte Datendatei in eine neue Datei und führt die notwendigen Konvertierungen durch:

```
' Definiere Verbunde für alten und neuen Dateipuffer:
TYPE AltPuffer
   MessReihe AS STRING*20
   XPos AS STRING*4
   YPos AS STRING*4
   FunkWert AS STRING*8
END TYPE

TYPE NeuPuffer
   MessReihe AS STRING*20
   XPos AS SINGLE
   YPos AS SINGLE
   FunkWert AS DOUBLE
END TYPE

' Deklariere Puffer:
DIM AltDatei AS AltPuffer, NeuDatei AS NeuPuffer
```

```
' Öffne alte und neue Datendatei:
OPEN "ALTMBF.DAT" FOR RANDOM AS #1 LEN=LEN(AltDatei)
OPEN "NEUIEEE.DAT" FOR RANDOM AS #2 LEN=LEN(NeuDatei)
I=1
' Lies den ersten alten Satz:
GET #1,I,AltDatei
DO UNTIL EOF(1)
' Bewege die Feldinhalte in die neuen Verbundfelder,
' konvertiere dabei die Inhalte von Feldern reeller
' Zahlen:
   NeuDatei.MessReihe=AltDatei.MessReihe
   NeuDatei.XPos=CVSMBF(AltDatei.XPos)
   NeuDatei.YPos=CVSMBF(AltDatei.YPos)
   NeuDatei.FunkWert=CVDMBF(AltDatei.FunkWert)
' Schreib die konvertierten Felder in die neue Datei:
   PUT #2,I,NeuDatei
   I=I+1
' Lies den nächsten Satz aus der alten Datei:
   GET #1,I,AltDatei
LOOP
CLOSE #1, #2
```

Jeder Satz der beiden Dateien hat vier Felder: ein Kennzeichenfeld, zwei Felder, die reelle Zahlen einfacher Genauigkeit enthalten, sowie ein letztes Feld, das eine reelle Zahl doppelter Genauigkeit enthält.

Die meiste Konvertierungsarbeit wird mit den Funktionen **CVDMBF** und **CVSMBF** erledigt. Die folgende Programmzeile konvertiert zum Beispiel das Feld doppelter Genauigkeit:

```
NeuDatei.FunkWert=CVDMBF(AltDatei.FunkWert)
```

Die acht Bytes des Verbundfeldes `AltDatei.FunkWert` werden mit der Funktion **CVDMBF** von einem Wert doppelter Genauigkeit im Microsoft-Binär-Format in einen Wert doppelter Genauigkeit im IEEE-Format umgewandelt. Dieser Wert wird in dem entsprechenden Feld des neuen Verbundes abgelegt, der später von der Anweisung **PUT** in die neue Datei geschrieben wird.

Weitere Informationen zu **CVDMBF** und verwandten Funktionen finden Sie in den entsprechenden Abschnitten im *BASIC-Befehlsverzeichnis.*

B.1.3 Direkte (on-line) Hilfe

QuickBASIC Version 4.0 enthält verschiedene Arten direkter Hilfe, die allgemeine Hilfe, Hilfe zu Menübefehlen sowie kontext-sensitive Hilfe zur BASIC-Syntax umfassen.

B.1.4 Lange (32-Bit-) Ganzzahlen

Lange (32-Bit) Ganzzahlen beseitigen Rundungsfehler in Berechnungen mit Zahlen, die in dem Bereich -2.147.483.648 bis 2.147.483.647 liegen, und sie bieten in diesem Bereich erheblich schnellere ganzzahlige Berechnungen, als dies mit Gleitkommazahlen möglich ist.

B.1.5 Zeichenketten fester Länge

Zeichenketten fester Länge ermöglichen es Ihnen, Zeichenkettendaten in Ihre benutzerdefinierten Typen einzubinden. Weitere Informationen finden Sie in Kapitel 2, "Datentypen", im *BASIC-Befehlsverzeichnis* und Kapitel 4, "Zeichenketten-Verarbeitung", in *Programmieren in BASIC: Ausgewählte Themen.*

B.1.6 Syntaxprüfung bei der Eingabe

Wenn die Syntaxüberprüfung eingeschaltet ist, überprüft QuickBASIC jede Zeile auf Syntax-, doppelte Definitions- sowie Speicherkapazitäts-Fehler, sobald Sie die Zeile abschließen. Eine Erläuterung der Syntaxüberprüfung beim Eingang sowie anderer Eigenschaften des "intelligenten" Editors finden Sie in Kapitel 5, "Bearbeiten".

B.1.7 Binär-Datei-E/A

Version 4.0 bietet binären Zugriff auf Dateien. Dies ist hilfreich für das Lesen und Verändern von Dateien, die in einem Nicht-ASCII-Format abgespeichert sind. Der Hauptvorteil von Binärzugriff liegt darin, daß Sie nicht gezwungen sind, mit einer Datei als Zusammenstellung von Sätzen umzugehen. Wenn eine Datei im Binärmodus geöffnet ist, können Sie sich vorwärts bzw. rückwärts auf jede beliebige Byte-Position in der Datei bewegen und anschließend so viele Bytes lesen oder schreiben, wie Sie möchten. Daher können unterschiedliche E/A-Operationen mit derselben Datei eine variierende Anzahl an Bytes lesen bzw. schreiben - anders als bei Direktzugriff, bei dem die Anzahl an Bytes festgelegt ist mit der definierten Länge eines einzelnen Satzes.

Weitere Informationen über den Zugriff auf Binär-Dateien finden Sie in Kapitel 3, "Datei- und Geräte-E/A", in *Programmieren in BASIC: Ausgewählte Themen.*

B.1.8 FUNCTION-Prozeduren

FUNCTION-Prozeduren ermöglichen es Ihnen, eine Funktion in einem Modul abzulegen und sie aus einem anderen Modul heraus aufzurufen. Weitere Informationen zur Verwendung von **FUNCTION**-Prozeduren finden Sie in Kapitel 2, "Prozeduren: Unterprogramme und Funktionen", in *Programmieren in BASIC: Ausgewählte Themen* sowie im *BASIC-Befehlsverzeichnis*.

B.1.9 Unterstützung für den CodeView-Debugger

Sie können die Befehlszeilen-Hilfsprogramme BC.EXE und LINK.EXE dazu einsetzen, ausführbare Dateien anzulegen, die kompatibel sind zu dem Microsoft CodeView-Debugger, einem leistungsfähigen Hilfsprogramm, das mit dem Microsoft Macro Assembler (Version 5.0) bzw. Microsoft C (Version 5.0) geliefert wird. Module, die mit BC kompiliert sind, können so mit Modulen, die mit anderen Microsoft-Sprachen kompiliert sind, gebunden werden, daß die schließlich ausführbare Datei mit dem CodeView-Debugger untersucht werden kann. Weitere Informationen finden Sie in Kapitel 9, "Wie Sie aus DOS heraus kompilieren und binden", sowie in Anhang C, "Wie Sie C- und Assembler-Routinen aufrufen".

B.1.10 Kompatibilität zu anderen Sprachen

QuickBASIC Version 4.0 ermöglicht es Ihnen, Routinen, die in anderen Microsoft-Sprachen geschrieben sind, mit den C- oder Pascal-Aufrufvereinbarungen aufzurufen. Argumente werden als kurze (NEAR) bzw. lange (FAR) Referenz oder als Wert übergeben. Anderssprachiger Code kann in einer Quick-Bibliothek abgelegt oder in ausführbaren Dateien gebunden werden.

B.1.11 Mehrere Module im Speicher

QuickBASIC Version 4.0 erlaubt es Ihnen, mehrere Programmodule gleichzeitig in den Speicher zu laden. Frühere QuickBASIC-Versionen erlaubten es nur einem Modul, im Speicher zu sein. Nun können Sie mehrmodulige Programme innerhalb der QuickBASIC-Umgebung bearbeiten, ausführen und debuggen. Weitere Informationen zur Verwendung mehrerer Module finden Sie in Kapitel 4, "Wie Sie Quelldateien verwalten", in diesem Handbuch und in Kapitel 2, "Prozeduren: Unterprogramme und Funktionen", in *Programmieren in BASIC: Ausgewählte Themen*.

B.1.12 Kompatibilität mit ProKey™, SideKick® und SuperKey®

Sie können ProKey, SideKick und SuperKey mit der Programm-Entwicklungsumgebung von Microsoft QuickBASIC Version 4.0 einsetzen. Andere die Tastatur umbelegende Software sowie Desktop-Software wird wahrscheinlich nicht mit QuickBASIC zusammen laufen. Sprechen Sie mit den Lieferanten oder Herstellern anderer Produkte, um in Erfahrung zu bringen, inwieweit das Produkt kompatibel zu QuickBASIC Version 4.0 ist.

B.1.13 Einfüge-/Überschreibemodus

Das Betätigen der EINFG-TASTE schaltet den Editiermodus zwischen Einfügen und Überschreiben um. Wenn der Überschreibemodus eingeschaltet ist, wechselt der Cursor von einem blinkenden Unterstreichungszeichen zu einem Rechteck. Beachten Sie, daß EINFG den STRG+O Einfüge-/Überschreibeschalter aus Version 3.0 ersetzt.

Im Einfügemodus fügt der Editor ein eingetipptes Zeichen an der Cursorposition ein. Im Überschreibemodus ersetzt der Editor das Zeichen unter dem Cursor mit dem Zeichen, das Sie eintippen. Der Einfügemodus ist der Standardmodus.

B.1.14 Tastaturbefehle im WordStar®-Stil

QuickBASIC unterstützt viele Tastenfolgen, mit denen Benutzer von WordStar vertraut sind. Eine komplette Liste der Tastaturbefehle im WordStar-Stil finden Sie in Kapitel 5, "Bearbeiten", Anhang D, "Zusammenfassung der Befehle" und auf der QuickBASIC 4.0 Bedienungskarte.

B.1.15 Rekursion

QuickBASIC Version 4.0 unterstützt Rekursion, das heißt die Fähigkeit, daß eine Prozedur sich selbst aufrufen kann. Rekursion ist hilfreich beim Lösen bestimmter Probleme, wie zum Beispiel Sortieren. Weitere Informationen zu der Verwendung von Rekursion in QuickBASIC-Programmen finden Sie in Kapitel 4, "Programme und Module", im *BASIC-Befehlsverzeichnis* sowie in Kapitel 2, "Prozeduren: Unterprogramme und Funktionen", in *Programmieren in BASIC: Ausgewählte Themen.*

B.1.16 Fehler-Listings während separater Kompilierung

QuickBASIC Version 4.0 zeigt nun erläuternde Fehlermeldungen an, wenn Sie Programme mit dem Befehl **bc** kompilieren. Weitere Informationen zu dem Befehl **bc** finden Sie in Kapitel 9, "Wie Sie aus DOS heraus kompilieren und binden".

Mit BC können Sie Fehlermeldungen auf eine Datei oder ein Gerät umleiten, um eine Kopie der Kompilierfehler zu erhalten.

Beispiele

Die folgenden Beispiele zeigen, wie die Befehlszeile **bc** zur Anzeige von Fehlern benutzt wird:

Befehlszeile	*Funktion*
`bc test.bas;`	Kompiliert die test.bas benannte Datei und zeigt Fehler auf dem Bildschirm an
`bc test.bas; > test.err`	Kompiliert die Datei test.bas und leitet die Fehlermeldungen in die test.err benannte Datei um.

B.1.17 Assembler-Listings während separater Kompilierung

Die Option **/a** des Befehls **bc** erstellt ein Listing des Assembler-Codes, der von dem Compiler erzeugt wird. Weitere Informationen zu dieser Option finden Sie in Abschnitt 9.4, "Wie Sie mit dem Befehl bc kompilieren".

B.2 Unterschiede in der Umgebung

Die Programmierumgebung von QuickBASIC 4.0 bietet flexiblere Wahlmöglichkeiten für Befehle, zusätzliche Fenster-Optionen sowie viel mehr Menübefehle, als das bei früheren QuickBASIC-Versionen der Fall ist. Die Abschnitte B.2.1 bis B.2.5 beschreiben die Unterschiede in der Programmierumgebung zwischen Version 4.0 und früheren Versionen.

B.2.1 Wie Sie Befehle und Optionen wählen

QuickBASIC Version 4.0 verhilft Ihnen zu größerer Flexibilität bei der Auswahl von Befehlen aus Menüs bzw. Optionen aus Dialogfeldern.

Version 4.0 erlaubt es Ihnen, jedes Menü zu wählen, indem Sie die ALT-Taste, gefolgt von einer mnemonischen Taste, betätigen. Jeder Menübefehl sowie jede Dialogfeld-Option haben ebenfalls jeweils eine mnemonische Taste, die einen Befehl sofort ausführt oder die Position sofort auswählt. Die mnemonische Taste erscheint auf einem Farbmonitor in rot, auf einer Kombination CGA-Adapter/Composite-Bildschirm hervorgehoben und auf einer Kombination Monochrom-Adapter/Monochrom-Bildschirm unterstrichen.

In Version 4.0 funktioniert die EINGABETASTE genauso wie die LEERTASTE in früheren Versionen. Sie können die EINGABETASTE betätigen, um einen Befehl aus einem Dialogfeld auszuführen.

B.2.2 Fenster

Version 4.0 erlaubt bis zu zwei Arbeitsfenster (bezeichnet als Arbeitsbereiche) und ein separates Direkt-Fenster. QuickBASIC Version 2.0 unterstützte nur zwei Fenster: den Arbeitsbereich sowie das Fenster für Fehlermeldungen. Weitere Informationen zu Fenstern finden Sie in Abschnitt 3.4.

B.2.3 Neue Menüs

QuickBASIC Version 4.0 verfügt über drei neue Menüs, die in der folgenden Tabelle aufgelistet sind.

Menü	*Beschreibung*
Debug	Ersetzt die Debug-Befehle aus Version 3.0. Weitere Informationen zu dem Menü **Debug** finden Sie in Kapitel 7, "Debuggen während der Programmierung".
Aufrufe	Zeigt die Folge von Aufrufen an, die innerhalb anderer Aufrufe verschachtelt sind. Weitere Informationen zu dem Menü **Aufrufe** finden Sie in Abschnitt 7.4.1.
Hilfe	Bietet Zugriff auf Informationen zu QuickBASIC-Editier- und -Menübefehlen sowie die BASIC-Syntax. Weitere Informationen zu dem Menü **Hilfe** finden Sie in Abschnitt 2.4.5.

B.2.4 Menübefehle

Die Menüs **Datei, Bearbeiten, Ansicht** und **Ausführen**, die bereits aus früheren QuickBASIC-Versionen bekannt sind, haben alle neue Befehle. Die folgende Tabelle führt die neuen Befehle dieser Menüs von QuickBASIC Version 4.0 auf.

Menü	*Befehl*	*Beschreibung*
Datei	**Neues Programm**	Identisch mit Neu in Version 2.0
	Programm laden	Identisch mit Laden... in Version 2.0
	Zusammenführen	Fügt eine Datei vor dem Cursor im aktiven Fenster (siehe Abschnitt 4.7) ein
	Speichern unter	Benennt und speichert ein noch unbenanntes Programm
	Alles Speichern	Ersetzt Autom. Speichern aus Version 2.0 (siehe Abschnitt 4.2.2)
	Datei anlegen	Legt ein neues Modul, eine neue Include-Datei oder ein neues Dokument an (siehe Abschnitt 4.3.1)
	Datei laden	Lädt ein Modul, eine Include-Datei oder ein Dokument von Diskette (siehe Abschnitt 4.3.1)
	Datei entfernen	Entfernt ein Modul, eine Include-Datei oder ein Dokument aus dem Speicher (siehe Abschnitt 4.4.2)
	Betriebssystem	Identisch mit Betriebssystem in Version 2.0
	Ende	Identisch mit Quit in Version 2.0
Bearbeiten	**Rückgängig**	Tastenkurzkombination ist nun ALT+RÜCKTASTE
	Ausschneiden	Tastenkurzkombination ist nun UMSCHALTTASTE+ENTF
	Kopieren	Tastenkurzkombination ist nun STRG+EINFG
	Einfügen	Tastenkurzkombination ist nun UMSCHALTTASTE+EINFG

Menü	*Befehl*	*Beschreibung*
Bearbeiten	**Löschen**	Löscht, ohne in die Zwischenablage zu kopieren (siehe Abschnitt 5.5.2)
	Neue SUB	Erzeugt neue **SUB**-Prozeduren (siehe Abschnitt 6.3.1)
	Neue FUNCTION	Erzeugt neue **FUNCTION**-Prozeduren (siehe Abschnitt 6.3.1)
	Syntax überprüfen	Schaltet die automatische Syntaxüberprüfung ein bzw. aus (siehe Abschnitt 5.2.3)
Ansicht	**SUBs**	Zeigt die aktuell geladenen Module und Prozeduren. Editiert, bewegt und löscht Module und Prozeduren (siehe Abschnitt 6.3.3)
	Nächste SUB	Blättert durch geladene Unterprogramme und **FUNCTION**-Prozeduren des aktuellen Moduls (siehe Abschnitt 6.3.3)
	Teilen	Öffnet oder schließt einen zweiten Arbeitsbereich (siehe Abschnitt 3.4.2)
	Nächste Anweisung	Bewegt den Cursor auf die nächste auszuführende Anweisung (siehe Abschnitt 7.3.5.4)
	Ausgabebildschirm	Betrachten der Programmausgabe (siehe Abschnitt 6.1.4)
	Bearbeiten Include-Datei	Editiert Include-Datei (siehe Abschnitt 4.5.2)
	Anzeigen Include-Datei	Betrachten der Include-Datei (siehe Abschnitt 4.5.2)
	Optionen	Erweitertes Dialogfeld zur Bedienung der neuen Bildschirmelemente der Version 4.0
Ausführen	**Start**	Identisch mit Version 2.0
	Neustart	Setzt Variablenwerte zurück als Vorbereitung für den Neustart im Einzelschrittmodus (siehe Abschnitt 7.3.5.5)
	Weiter	Setzt die Ausführung fort nach einer Unterbrechung der Ausführung (siehe Abschnitt 7.3.5.1)

Menü	*Befehl*	*Beschreibung*
Ausführen	**Ändere COMMAND$**	Setzt die von **COMMAND$** gelesene Zeichenkette (siehe Abschnitt 6.1.6)
	EXE-Datei erstellen	Erzeugt eine ausführbare Datei (siehe Abschnitt 6.2)
	Bibliothek erstellen	Erzeugt eine Quick-Bibliothek (siehe Abschnitt 8.3.3)
	Hauptmodul bestimmen	Legt ein neues Hauptmodul fest (siehe Abschnitt 4.4)

Die folgenden Menübefehle von Version 2.0 sind aus Version 4.0 entfernt:

- Fehler (Menü Ansicht)

  QuickBASIC-Version 4.0 zeigt Fehlermeldungen an, während Sie das Programm eingeben oder ausführen.
- Nächster Fehler (Menü Suchen)

  Da Fehler während der Eingabe oder der Ausführung des Programms bemerkt werden, besteht keine Notwendigkeit, nach der Fehlerposition zu suchen.
- Kompilieren (Menü Betrieb)

  Der Befehl Kompilieren ist durch den Befehl **Start** aus dem Menü **Ausführen** ersetzt, da Ihr Programm immer in der QuickBASIC-Umgebung ausgeführt werden kann.
- Kompilieren... (Menü Betrieb)

  Der Befehl Kompilieren... ist durch den Befehl **EXE-Datei erstellen** aus dem Menü **Ausführen** ersetzt. Es ist nicht länger notwendig, Compiler-Optionen zu setzen, wenn im Speicher ausgeführt wird, oder wenn mit dem Befehl **EXE-Datei erstellen** eine ausführbare Datei erzeugt wird.

Die Option "Debug" der Version 3.0 ist aus dem Menü **Ausführen** entfernt. In Version 4.0 können Sie jederzeit mit den Debug-Befehlen des Menüs **Debug** Fehler entfernen.

B.2.5 Änderungen der Tasten zur Bearbeitung des Programmtextes

Die QuickBASIC Version 4.0 Tastaturschnittstelle ist so erweitert, daß sie Tastenkombinationen zur Bearbeitung des Programmtextes enthält, die denen des WordStar-Editors ähnlich sind. Eine Liste aller QuickBASIC-Befehle zur Bearbeitung des Programmtextes finden Sie in Abschnitt D.4. Die Funktionen, die von den in der folgenden Tabelle aufgeführten Tastenfolgen in QuickBASIC-Version 2.0 durchgeführt werden, sind in QuickBASIC-Version 4.0 geändert.

Funktion	*QuickBASIC 2.0*	*QuickBASIC 4.0*
Rückgängig	UMSCHALTTASTE+ESC	ALT+RÜCKTASTE
Ausschneiden	ENTF	UMSCHALTTASTE+ENTF
Kopieren	F2	STRG+EINFG
Einfügen	EINFG	UMSCHALTTASTE+EINFG
Löschen	---	ENTF
Überschreiben	---	EINFG

B.3 Unterschiede beim Debuggen und Kompilieren

In der QuickBASIC-Programmierumgebung der Version 4.0 sind das Kompilieren und das Debuggen keine separaten Operationen. Ihr Programm ist jederzeit ausführbar, und Sie können Ihren Code auf verschiedene Weise während der Programmierung von Fehlern befreien. Dieser Abschnitt beschreibt die Unterschiede der Kompilier- und Debug-Eigenschaften zwischen QuickBASIC Version 4.0 und früheren Versionen.

B.3.1 Unterschiede der Befehlszeile

QuickBASIC Version 4.0 unterstützt Befehlszeilen-Optionen der Version 2.0 nur für den Befehl **qb**, nicht für den Befehl **bc**. Um ein Programm außerhalb der QuickBASIC-Umgebung zu kompilieren, verwenden Sie den Befehl **bc**, der in Abschnitt 9.4 beschrieben ist.

Version 4.0 erfordert keine der Kompilier-Optionen, die in Tabelle 4.1 des Handbuches zu Microsoft QuickBASIC Version 2.0 aufgeführt sind. Wenn Sie versuchen, QuickBASIC mit diesen Optionen als Befehlszeilen-Optionen aufzurufen, erscheint eine Fehlermeldung. Ähnlich ist es in Version 3.0 notwendig, bestimmte Kompilier-Optionen aus dem Dialogfeld "Compile" zu wählen; dies wird nun automatisch von Version 4.0 erledigt. Die folgende Tabelle beschreibt die Art und Weise, in der QuickBASIC Version 4.0 die Funktionen dieser Optionen unterstützt.

Version 2.0	*Version 4.0*
On Error (/e)	Automatisch gesetzt, wenn eine **ON ERROR**-Anweisung vorhanden ist.
Debug (/d)	Immer eingeschaltet, wenn Sie ein Programm innerhalb der QuickBASIC-Umgebung starten. Wenn Sie ausführbare Programme auf Diskette oder in Quick-Bibliotheken anlegen, verwenden Sie die Option "Debug-Code erstellen".

Version 2.0	*Version 4.0*
Zwischen Anweisungen prüfen (/v) und Ereigniserfassung (/w)	Automatisch gesetzt, wenn eine **ON** *Ereignis*-Anweisung vorhanden ist.
Resume Next (/x)	Automatisch gesetzt, wenn eine **RESUME NEXT**-Anweisung vorhanden ist.
Datenfelder in Zeilenanordnung (/r)	Verfügbar nur bei Kompilierung mit **bc**.
Zeichenkettendaten minimieren (/s)	Der Standard für **qb**. Um die Option "Zeichenkettendaten minimieren" auszuschalten, müssen Sie mit **bc** von der Befehlszeile aus kompilieren.

Die Optionen, die in der folgenden Tabelle aufgeführt sind, sind nun für die **qb**- und **bc**-Befehle verfügbar:

Option	*Beschreibung*
/ah	Ermöglicht es dynamischen Verbunddatenfeldern, Zeichenketten fester Länge sowie numerischen Daten, jeweils größer als 64K zu sein. Falls diese Option nicht angegeben ist, beträgt die maximale Größe pro Datenfeld 64K. Beachten Sie, daß diese Option keine Auswirkung auf die Art hat, mit der Datengrößen an Prozeduren übergeben werden. (Diese Option wird mit den Befehlen **qb** und **bc** verwendet.)
/h	Zeigt mit der auf Ihrer Hardware höchstmöglichen Auflösung an. Wenn Sie zum Beispiel eine EGA haben, zeigt QuickBASIC Text mit 43 Zeilen und 80 Spalten an. (Diese Option wird nur mit dem Befehl **qb** verwendet.)
/mbf	Konvertiert Zahlen im Microsoft-Binär-Format in das IEEE-Format. Weitere Informationen zu dieser Option finden Sie in Abschnitt B.1.2.3. (Diese Option wird mit den Befehlen **qb** und **bc** verwendet.)
/run *Quelldatei*	Startet *Quelldatei* sofort, ohne zunächst die QuickBASIC-Programmierumgebung anzuzeigen. (Diese Option wird nur mit dem Befehl **qb** verwendet.)

B.3.2 Unterschiede bei separater Kompilierung

Version 4.0 läßt keine separate Kompilierung mit dem Befehl **qb** zu. Verwenden Sie den in Abschnitt 9.4 beschriebenen Befehl **bc**, um Dateien zu kompilieren und binden, ohne die Programmierumgebung aufzurufen.

B.3.3 Benutzerbibliotheken und BUILDLIB

Benutzerbibliotheken, die für frühere Versionen erstellt wurden, sind nicht kompatibel zu Version 4.0. Sie müssen die Bibliothek aus den ursprünglichen Quelldateien neu aufbauen.

In Version 4.0 werden Benutzerbibliotheken als Quick-Bibliotheken bezeichnet. Es gibt keinen Unterschied in ihrer Funktion oder Verwendung. Die Erweiterung des Dateinamens dieser Bibliotheken ist nun .QLB anstelle von .EXE. Das Hilfsprogramm BUILDLIB wird nicht länger benötigt. Quick-Bibliotheken werden nun aus der Programmierumgebung heraus oder von der **link**-Befehlszeile aus angelegt. Weitere Informationen finden Sie in Kapitel 8, "Quick-Bibliotheken".

B.3.4 Einschränkungen für Include-Dateien

Include-Dateien dürfen keine **SUB**- oder **FUNCTION**-Prozedurdefinitionen enthalten. Verwenden Sie in solchen Fällen den Befehl **Zusammenführen** aus dem Menü **Datei**, um die Include-Datei in das aktuelle Modul einzufügen. Wenn Sie eine Include-Datei einfügen, die eine **SUB**-Prozedur enthält, erscheint der Text der Prozedur nicht im aktuell aktiven Fenster. Um den Text des Fensters zu betrachten oder zu bearbeiten, betätigen Sie F2 und wählen anschließend den Prozedurnamen aus dem Listenfeld. Nachdem der Text einmal eingemischt ist, können Sie das Programm starten.

Andererseits entschließen Sie sich vielleicht dazu, Ihre **SUB**-Prozedur in einem separaten Modul abzulegen. In diesem Fall müssen Sie einen der folgenden beiden Schritte für jede gemeinsam benutzte Variable (Variablen, die in einer **COMMON SHARED**- oder **[RE]DIM SHARED**-Anweisung außerhalb der **SUB**-Prozedur oder in einer **SHARED**-Anweisung innerhalb der **SUB**-Prozedur deklariert sind) durchführen, da Variablen, die auf diese Weise deklariert sind, nur innerhalb eines einzigen Moduls gemeinsam benutzt werden:

1. Benutzen Sie die Variablen zwischen den Modulen gemeinsam, indem Sie die Variablen in beiden Modulen auf Modul-Ebene in **COMMON**-Anweisungen aufführen.
2. Übergeben Sie die Variablen an die **SUB**-Prozedur in einer Argumentenliste.

Eine detaillierte Erklärung zu Include-Dateien und Modulen in QuickBASIC finden Sie in Kapitel 4, "Wie Sie Quelldateien verwalten". In Kapitel 2, "Prozeduren: Unterprogramme und Funktionen", in *Programmieren in BASIC: Ausgewählte Themen* und in Kapitel 4, "Programme und Module", im *BASIC-Befehlsverzeichnis* finden Sie zusätzliche Informationen zu Modulen und Prozeduren.

B.3.5 Debuggen

QuickBASIC-Version 4.0 unterstützt Sie dabei, Ihre Programme schneller von Fehlern zu befreien, indem es die folgenden Debug-Eigenschaften bietet:

- Mehrere Haltepunkte
- Anzeigeausdrücke und Stoppbedingungen
- Verbesserte Programmverfolgung
- Ein Bildschirmfenster, das während des Einzelschrittmodus Quellebenen-Programmtext anzeigt
- Die Fähigkeit, Variablenwerte während der Ausführung zu ändern, und die Ausführung anschließend fortzusetzen
- Die Fähigkeit, Programme zu bearbeiten und anschließend mit der Ausführung fortzufahren, ohne neu zu starten
- Rückverfolgung
- Menü **Aufrufe**

Die in der folgenden Tabelle aufgeführten Funktionstastenkombinationen zur Fehlersuche haben sich in QuickBASIC-Version 4.0 geändert:

Funktion	***QuickBASIC Version 2.0***	***QuickBASIC Version 4.0***
Verfolgen	ALT+F8	F8
Schritt	ALT+F9	F10

Beachten Sie, daß der Animationsmodus eingeschaltet ist, wenn Sie den Befehl **Verfolgen ein** aus dem Menü **Debug** einschalten und anschließend das Programm starten.

Weitere Informationen finden Sie in Kapitel 7, "Debuggen während der Programmierung".

B.4 Änderungen an der Sprache BASIC

Dieser Abschnitt beschreibt Erweiterungen und Änderungen der Sprache BASIC in Version 4.0 sowie früheren QuickBASIC-Versionen. Die folgende Tabelle führt die Schlüsselwörter auf, die von diesen Änderungen betroffen sind, und zeigt, welche QuickBASIC-Version von jeder Änderung betroffen ist. Ausführlichere Erläuterungen dieser Änderungen finden Sie im Anschluß an die folgende Tabelle. Umfassende Informationen über die Sprache BASIC finden Sie im *BASIC-Befehlsverzeichnis*.

Schlüsselwort	*QuickBASIC0 Version 2.0*	*QuickBASIC Version 3.0*	*QuickBASIC Version 4.0*
AS	Nein	Nein	Ja
CALL	Nein	Nein	Ja
CASE	Nein	Ja	Ja
CLEAR	Nein	Nein	Ja
CLNG	Nein	Nein	Ja
CLS	Nein	Ja	Ja
COLOR	Nein	Nein	Ja
CONST	Nein	Ja	Ja
CVL	Nein	Nein	Ja
CVSMBF, CVDMBF	Nein	Ja	Ja
DECLARE	Nein	Nein	Ja
DEFLNG	Nein	Nein	Ja
DIM	Nein	Nein	Ja
DO...LOOP	Nein	Ja	Ja
EXIT	Nein	Ja	Ja
FILEATTR	Nein	Nein	Ja
FREEFILE	Nein	Nein	Ja
FUNCTION	Nein	Nein	Ja
GET	Nein	Nein	Ja
LCASE$	Nein	Nein	Ja
LEN	Nein	Nein	Ja
LSET	Nein	Nein	Ja
LTRIM$	Nein	Nein	Ja
MKL$	Nein	Nein	Ja
MKSMBF$, MKDMBF$	Nein	Ja	Ja
OPEN	Nein	Nein	Ja
PALETTE	Nein	Nein	Ja
PUT	Nein	Nein	Ja

Schlüsselwort	*QuickBASIC0 Version 2.0*	*QuickBASIC Version 3.0*	*QuickBASIC Version 4.0*
RTRIM$	Nein	Nein	Ja
SCREEN	Nein	Nein	Ja
SEEK	Nein	Nein	Ja
SELECT CASE	Nein	Ja	Ja
SETMEM	Nein	Nein	Ja
STATIC	Nein	Nein	Ja
TYPE	Nein	Nein	Ja
UCASE$	Nein	Nein	Ja
VARPTR	Nein	Nein	Ja
VARSEG	Nein	Nein	Ja
WIDTH	Nein	Ja	Ja

Der folgende Abschnitt erläutert ausführlicher die Unterschiede der Schlüsselwörter, die in obiger Tabelle zusammengefaßt sind.

Schlüsselwörter	*Erklärung*
AS	Die Klausel **AS** ermöglicht die Verwendung benutzerdefinierter Typen in **DIM**-, **COMMON**- und **SHARED**-Anweisungen, sowie in **DECLARE**-, **SUB**- und **FUNCTION**-Parameterlisten.
CALL	Die Verwendung von **CALL** ist optional für den Aufruf von Unterprogrammen, die mit der Anweisung **DECLARE** deklariert sind.
CLEAR	Die Anweisung **CLEAR** setzt nicht länger die Gesamtgröße des Stapels. In Version 4.0 setzt die Anweisung **CLEAR** nur die von dem Programm benötigte Stapelgröße. QuickBASIC setzt die Stapelgröße auf den Betrag, der von der **CLEAR**-Anweisung angegeben wird, plus den Betrag, den QuickBASIC selbst benötigt.
CLNG	Die Funktion **CLNG** rundet ihr Argument und gibt eine lange (4-Byte) Ganzzahl zurück, die dem Argument entspricht.
CLS	Die Anweisung **CLS** wurde verändert, um Ihnen eine größere Beweglichkeit beim Löschen des Bildschirms zu geben. Beachten Sie, daß QuickBASIC den Bildschirm am Beginn jedes Programms nicht mehr automatisch löscht, wie das in früheren Versionen der Fall war.

Schlüsselwörter	*Erklärung*
COLOR, SCREEN, PALETTE, WIDTH	Die Anweisungen **COLOR, SCREEN, PALETTE, WIDTH** sind erweitert, um die Bildschirmmodi einzuführen, die mit den IBM PS/2 VGA und Multicolor Graphics Array (MCGA) verfügbar sind.
CONST	**CONST**-Anweisungen ermöglichen Ihnen die Definition symbolischer Konstanten, um die Lesbarkeit und Wartbarkeit Ihrer Programme zu verbessern.
CVL	Die Funktion **CVL** wird dazu verwendet, lange Ganzzahlen zu lesen, die in Direktzugriffs-Datendateien als Zeichenketten gespeichert sind. **CVL** konvertiert eine Vier-Byte-Zeichenkette, die mit der Funktion **MKL$** erzeugt wurde, für die Verwendung in Ihrem BASIC-Programm zurück in eine lange Ganzzahl.
CVSMBF, CVDMBF	Die Funktionen **CVSMBF, CVDMBF** konvertieren Zeichenketten, die Zahlen im Microsoft-Binär-Format enthalten, in Zahlen im IEEE-Format.
DECLARE	Die Anweisung **DECLARE** ermöglicht es Ihnen, Prozeduren aus unterschiedlichen Modulen heraus aufzurufen, Anzahl und Typ der übergebenen Argumente zu überprüfen sowie Prozeduren aufzurufen, bevor sie definiert sind.
DEFLNG	Die Anweisung **DEFLNG** deklariert alle Variablen, **DEF FN**-Funktionen und **FUNCTION**-Prozeduren so, daß sie den Typ lange Ganzzahl haben. Das heißt, eine Variable oder Funktion ist standardmäßig eine lange Ganzzahl, es sei denn, sie wurde in einer Klausel **AS** *Typ* deklariert, oder sie hat ein ausdrückliches Typdefinitionssuffix, wie zum Beispiel % oder $.
DIM	Die Klausel **TO** der Anweisung **DIM** ermöglicht es Ihnen, Indizes mit beliebigen ganzzahligen Werten anzugeben und gibt Ihnen damit größere Beweglichkeit bei Datenfelddeklarationen.
DO...LOOP	**DO...LOOP**-Anweisungen stellen Ihnen leistungsfähigere Schleifen zur Verfügung, die es Ihnen ermöglichen, besser strukturierte Programme zu schreiben.
EXIT	**EXIT {DEF | DO | FOR | FUNCTION | SUB}**-Anweisungen bieten bequeme Ausstiegsmöglichkeiten aus Schleifen und Prozeduren.
FREEFILE, FILEATTR	Die Funktionen **FREEFILE** und **FILEATTR** helfen Ihnen, Anwendungen zu schreiben, die Datei-E/A in einer mehrmoduligen Umgebung erledigen.

Schlüsselwörter	*Erklärung*
FUNCTION	Die **FUNCTION...END FUNCTION**-Prozedur ermöglicht es Ihnen, eine mehrzeilige Prozedur zu definieren, die Sie aus einem Ausdruck heraus aufrufen können. Diese Prozeduren verhalten sich im wesentlichen wie eingebaute Funktionen, zum Beispiel wie **ABS** oder mehrzeilige **DEF FN**-Funktionen der QuickBASIC-Versionen 1.0 bis 3.0. Anders als eine **DEF FN**-Funktion kann eine **FUNCTION**-Prozedur jedoch in einem Modul definiert sein und aus einem anderen heraus aufgerufen werden. Außerdem kennen **FUNCTION**-Prozeduren lokale Variablen und unterstützen Rekursionen.
GET, PUT	Für E/A-Operationen ist die Syntax der **GET**- bzw. **PUT**-Anweisung erweitert, um mit **TYPE...END TYPE**-Anweisungen definierte Verbunde einzubeziehen. Dies macht die Verwendung der **FIELD**-Anweisung überflüssig.
LCASE$, UCASE$, LTRIM$, RTRIM$	Die folgenden Funktionen zur Zeichenkettenbehandlung sind in Version 4.0 verfügbar:

Funktion	*Rückgabewert*
LCASE$	Eine Kopie der Zeichenkette, wobei alle Buchstaben in Kleinbuchstaben umgewandelt sind
UCASE$	Eine Kopie der Zeichenkette, wobei alle Buchstaben in Großbuchstaben umgewandelt sind
LTRIM$	Eine Kopie der Zeichenkette, wobei alle führenden Leerzeichen entfernt sind
RTRIM$	Eine Kopie der Zeichenkette, wobei alle nachfolgenden Leerzeichen entfernt sind

LEN	Die Funktion **LEN** ist so erweitert, daß sie die Anzahl an Bytes zurückgibt, die eine skalare oder Verbundvariable, eine Konstante, ein Ausdruck oder ein Datenfeldelement benötigt.
LSET	Die Anweisung **LSET** ist erweitert, um sowohl Verbund- als auch Zeichenkettenvariablen einzubeziehen. Dies ermöglicht es Ihnen, eine Verbundvariable einer anderen Verbundvariablen zuzuweisen, selbst wenn die Verbunde nicht gleich sind.

Schlüsselwörter	*Erklärung*
MKL$	Die Funktion **MKL$** wird dazu eingesetzt, lange Ganzzahlen in Zeichenketten umzuwandeln, die in Direktzugriffs-Datendateien gespeichert werden können. Verwenden Sie die Funktion **CVL**, um die Zeichenkette in eine lange Ganzzahl zurückzuwandeln.
MKSMBF$, MKDMBF$	Die Funktionen **MKSMBF$** und **MKDMBF$** wandeln Zahlen im IEEE-Format in Zeichenketten um, die Zahlen im Microsoft-Binär-Format enthalten. Dies ermöglicht es Ihnen, mit der Version 4.0 Datendateien einzusetzen, die für Vorgängerversionen von 4.0 angelegt wurden.

OPEN	Die Anweisung **OPEN** öffnet nun zwei Dateien desselben Namens für **OUTPUT** oder **APPEND**, solange sich die Pfadnamen unterscheiden. Zum Beispiel ist nun folgendes erlaubt:
	
	`OPEN "BEISP" FOR APPEND AS #1`
	`OPEN "TMP\BEISP" FOR APPEND AS #2`
	
	Der Syntax der **OPEN**-Anweisung ist ein binärer Dateimodus hinzugefügt worden. Informationen zur Verwendung dieses Modus finden Sie in Kapitel 3, "Datei- und Geräte-E/A", in *Programmieren in BASIC: Ausgewählte Themen.*

SEEK	Die **SEEK**- Anweisung und -Funktion ermöglichen es Ihnen, eine Datei auf irgendein Byte oder irgendeinen Satz zu positionieren. Weitere Informationen finden Sie in Kapitel 3, "Datei- und Geräte-E/A", in *Programmieren in BASIC: Ausgewählte Themen* sowie in den Beschreibungen für die **SEEK**-Anweisung und -Funktion im *BASIC-Befehlsverzeichnis.*
SELECT CASE	**SELECT CASE**-Anweisungen bieten eine Möglichkeit zur Vereinfachung komplexer Abfragebedingungen. Die **CASE**-Klauseln der **SELECT CASE**-Anweisung akzeptieren jetzt jeden Ausdruck (einschließlich variabler Ausdrücke) als Argument; in früheren Versionen waren nur konstante Ausdrücke erlaubt.
SETMEM	Die Funktion **SETMEM** erleichtert mehrsprachige Programmierung, indem Sie es Ihnen ermöglicht, den von BASIC zugewiesenen dynamischen Speicherbetrag zu verkleinern, so daß dieser von anderssprachigen Prozeduren verwendet werden kann.
STATIC	Das Auslassen des **STATIC**-Attributs aus **SUB**- und **FUNCTION**-Anweisungen veranlaßt, daß Variablen beim Aufruf der Prozeduren zugewiesen werden und nicht bei der Definition der Prozeduren. Solche Variablen behalten ihre Werte zwischen Prozeduraufrufen nicht.

Schlüsselwörter	*Erklärung*
TYPE	Die Anweisung **TYPE...END TYPE** ermöglicht es Ihnen, Datentypen zu definieren, die Elemente unterschiedlicher einfacher Typen enthalten. Dies vereinfacht die Definierung von sowie den Zugriff auf Sätzen in Direktzugriffsdateien.
VARPTR	Die Funktion **VARPTR** gibt nun den ganzzahligen 16-Bit-Offset der BASIC-Variablen oder des BASIC-Datenfeldelementes zurück. Der Offset beginnt am Anfang des Segmentes, das die Variable oder das Datenfeldelement enthält.
VARSEG	Die Funktion **VARSEG** gibt die Segmentadresse ihres Argumentes zurück. Dies ermöglicht es Ihnen, **DEF SEG** passend zur Verwendung mit **PEEK, POKE, BLOAD, BSAVE** sowie **CALL ABSOLUTE** zu setzen. Außerdem ermöglicht es Ihnen die Funktion, das passende Segment für die Verwendung mit **CALL** INTERRUPT zu erhalten, wenn Betriebssystem- oder BIOS-Interrupts ausgeführt werden.
WIDTH	Ein neues Argument der Anweisung **WIDTH** ermöglicht es Ihren Programmen, die erweiterten Zeilenmodi auf Maschinen zu verwenden, die mit einer EGA-, VGA- oder MCGA-Adapterkarte ausgerüstet sind.

Wichtig Mit der einzeiligen **IF...THEN...ELSE**-Anweisung dürfen Sie nicht länger **NEXT**- und **WEND**-Anweisungen bedingt ausführen. Weitere Informationen finden Sie in der Beschreibung der einzeiligen **IF...THEN...ELSE**-Anweisung im *BASIC-Befehlsverzeichnis*.

B.5 Datei-Kompatibilität

Alle QuickBASIC-Versionen sind Quellcode-kompatibel; Quellcode, der für eine frühere Version angelegt wurde, wird von Version 4.0 kompiliert. Die Ausnahmen hiervon sind in Abschnitt B.3.4, "Einschränkungen für Include-Dateien", aufgeführt. QuickBASIC 4.0 ist jedoch *nicht* Binär-kompatibel zu früheren Versionen. Objektdateien und Benutzerbibliotheken, die für frühere QuickBASIC-Versionen angelegt wurden, müssen Sie erneut kompilieren.

Anhang C: Wie Sie C- und Assembler-Routinen aufrufen

Mehrsprachige Programmierung bezeichnet den Prozeß der Kombinierung von Programmen aus zwei oder mehreren Quellensprachen. Zum Beispiel erlaubt es Ihnen mehrsprachige Programmierung, Assembler einzusetzen, um Ihre QuickBASIC-Programme zu verbessern. Sie können den größten Teil Ihres Programmes schnell mit QuickBASIC entwickeln und dann Assembler für Routinen einsetzen, die häufig ausgeführt werden und mit höchster Geschwindigkeit laufen müssen. Ähnlich können Sie Ihre eigenen C-Funktionen aus QuickBASIC-Programmen heraus aufrufen, und genausogut können Sie viele der Routinen der Microsoft C-Standard-Bibliothek aufrufen.

In diesem Anhang wird vorausgesetzt, daß Sie die Sprachen, die Sie kombinieren möchten, kennen, und daß Sie wissen, wie mehrmodulige Programme mit diesen Sprachen geschrieben, kompiliert und gebunden werden.

Nachdem Sie diesen Anhang gelesen haben, werden Sie folgendes verstehen:

- Allgemeine Probleme, die bei mehrsprachiger Programmierung zu beachten sind.
- Zwischensprachliche Aufrufe zwischen BASIC, C und Assembler.
- Die Übergabe von Parametern in zwischensprachlichen Aufrufen.
- Die Unterschiede, wie BASIC bzw. C numerische und Zeichenketten-Daten behandeln.
- Die Verwendung von Strukturen und benutzerdefinierten Typen in Aufrufen zwischen C und BASIC.

Hinweis Die in diesem Anhang verwendeten Darstellungsvereinbarungen finden Sie in der "Einführung" zu *Programmieren in BASIC: Ausgewählte Themen.*

C.1 Wie Sie mehrsprachige Programme aufbauen

Die Art, in der Sie mehrsprachige Programme aufbauen, hängt davon ab, ob Sie Ihr Programm innerhalb der QuickBASIC-Programm-Entwicklungsumgebung laufen lassen, oder ob Sie es mit **BC** von der DOS-Befehlszeile aus kompilieren. Wenn Ihr Programm anderssprachige Routinen aus der QuickBASIC-Umgebung heraus aufruft, dann müssen die anderssprachigen Routinen zunächst kompiliert und in einer Quick-Bibliothek gebunden werden, und die Quick-Bibliothek muß mit QuickBASIC geladen werden.

Hinweis Es ist besonders wichtig, daß anderssprachige Prozeduren sorgfältig debugged sind, *bevor* sie in eine Quick-Bibliothek eingebunden werden. QuickBASIC "läuft" bei der Verfolgung durch ein Programm nicht in die Prozeduren der Quick-Bibliothek "hinein", so daß deren Debuggen aus der Umgebung heraus nicht möglich ist.

Der mit Microsoft C Version 5.0, Microsoft Macro Assembler (**MASM**) Version 5.0, Microsoft Pascal Version 4.0 und Microsoft FORTRAN Version 4.0 gelieferte Microsoft CodeView-Debugger erlauben es Ihnen, mehrsprachige Programme zu debuggen.

Falls Sie Ihr Programm von der DOS-Befehlszeile aus kompilieren und binden, müssen Ihre anderssprachigen Routinen nicht Teil einer Bibliothek sein. Wenn Sie dies jedoch angenehmer finden, wird für diesen Zweck der Microsoft Bibliotheks-Manager (**LIB**) geliefert. Weitere Informationen zum Kompilieren, Binden und Verwalten von Bibliotheken finden Sie in Kapitel 9, "Wie Sie aus DOS heraus kompilieren und binden".

C.2 Elemente mehrsprachiger Programmierung

Die Microsoft-Sprachen kennen besondere Schlüsselwörter, die mehrsprachige Programmierung unterstützen. Um diese Schlüsselwörter zu verwenden, müssen Sie die damit verknüpften Grundbegriffe verstehen.

Die Abschnitte C.2.1 bis C.2.4 beschreiben die Elemente mehrsprachiger Programmierung: wie sich Sprachen unterscheiden, und wie diese Unterschiede gelöst werden. Wenn Sie die in diesen Abschnitten beschriebenen Prinzipien verstehen, dann möchten Sie vielleicht direkt mit späteren Abschnitten dieses Anhangs fortfahren. Trotzdem werden Sie es eventuell hilfreich finden, sich gelegentlich auf diese Abschnitte zu beziehen.

Der Abschnitt C.2.1 stellt den Grundzusammenhang eines mehrsprachigen Aufrufs dar und beschreibt, wann und wie Sie einen solchen Aufruf durchführen.

Die Abschnitte C.2.2 bis C.2.4 stellen die drei grundlegenden Voraussetzungen für mehrsprachige Programmierung dar:

- Voraussetzung der Vereinbarungen zur Benennung
- Voraussetzung der Vereinbarungen für Aufrufe
- Voraussetzung zur Parameterübergabe

Der Abschnitt C.2.5 stellt die Kompilierzeit- sowie Bindezeit-Probleme einschließlich der Verwendung von Speichermodellen dar.

C.2.1 Wie Sie mehrsprachige Aufrufe erstellen

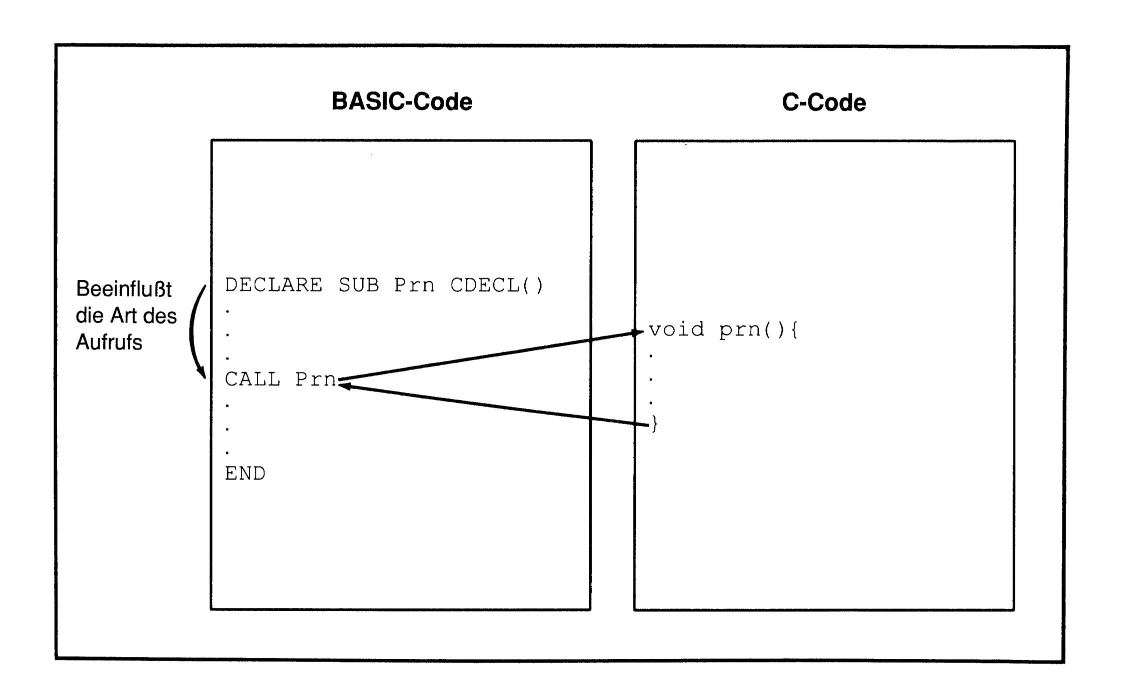
Mehrsprachige Programmierung umfaßt immer den Aufruf einer Funktion oder Prozedur. Zum Beispiel könnte ein BASIC-Hauptmodul eine spezielle Aufgabe ausführen müssen, die Sie separat programmieren möchten. Anstatt jedoch ein BASIC-Unterprogramm aufzurufen, entscheiden Sie sich, eine C-Funktion aufzurufen.

Mehrsprachige Aufrufe umfassen notwendigerweise mehrere Module. Anstatt alle Ihrer Quellmodule mit demselben Compiler zu kompilieren, verwenden Sie unterschiedliche Compiler. In dem oben erwähnten Beispiel könnten Sie die Quelldatei des Hauptmoduls mit **BC** kompilieren, eine andere Quelldatei (geschrieben in C) mit dem C-Compiler kompilieren und anschließend die beiden Objektdateien mit **LINK** binden. Alternativ dazu könnten Sie die C-Funktion kompilieren, anschließend in eine Quick-Bibliothek einbinden und die Funktion aus einem Programm heraus aufrufen, das innerhalb der QuickBASIC-Umgebung läuft.

Hinweis Objektdateien, die entweder mit dem Microsoft C Optimizing Compiler oder Microsoft QuickC erstellt sind, sind für den Einsatz in mehrsprachigen Programmen geeignet.

Abbildung C.1 veranschaulicht die Syntax eines mehrsprachigen Aufrufs, in dem ein BASIC-Hauptmodul eine C-Funktion aufruft.

Abbildung C.1 Mehrsprachiger Aufruf



In Abbildung C.1 lautet der BASIC-Aufruf zu C `CALL Prn`. Die Form ist ähnlich dem Aufruf eines BASIC-Unterprogrammes. Es gibt jedoch zwei Unterschiede zwischen diesem mehrsprachigen Aufruf und einem Aufruf zwischen zwei BASIC-Modulen:

1. Das Unterprogramm `Prn` ist eigentlich in C mit der C-Standardsyntax geschrieben.
2. Die Ausführung des Aufrufs in BASIC wird beeinflußt durch das Vorhandensein einer **DECLARE**-Anweisung, die das Schlüsselwort **CDECL** verwendet, um Kompatibilität mit C herzustellen.

Die **DECLARE**-Anweisung ist ein Beispiel für eine mehrsprachige "Schnittstellen"-Anweisung. Jede Sprache bietet eine eigene Schnittstellenform. Weitere Informationen finden Sie in diesem Handbuch in Abschnitt C.3.1.1, "Die Anweisung DECLARE"; in den Erläuterungen zur Anweisung **DECLARE** im *BASIC-Befehlsverzeichnis* sowie in *Programmieren in BASIC: Ausgewählte Themen.*

Trotz der syntaktischen Unterschiede sind QuickBASIC-**FUNCTION**- und -**SUB**-Prozeduren sehr ähnlich zu C-Funktionen und Assembler-Prozeduren. Der prinzipielle Unterschied besteht darin, daß alle C-Funktionen, QuickBASIC-**FUNCTION**-Prozeduren und Assembler-Prozeduren Werte zurückgeben können, während QuickBASIC-**SUB**-Prozeduren dies nicht können.

Hinweis In diesem Anhang bezieht sich "Routine" auf jede C-Funktion, QuickBASIC-**FUNCTION**- und -**SUB**-Prozedur oder Assembler-Prozedur, die von einem anderen Modul aufgerufen werden kann.

Die folgende Tabelle zeigt die Entsprechungen zwischen Aufrufen von Routinen in unterschiedlichen Sprachen.

Sprache	***Rückgabe eines Wertes***	***Keine Rückgabe eines Wertes***
BASIC	**FUNCTION**-Prozedur	**SUB**-Prozedur
C	Funktion	**(void)** Funktion
Macro-Assembler	Prozedur	Prozedur

Zum Beispiel kann ein BASIC-Modul einen **SUB**-Prozeduraufruf einer C-Funktion durchführen, die anstelle eines Rückgabetyps mit dem Schlüsselwort **void** deklariert ist. BASIC sollte einen **FUNCTION**-Prozeduraufruf ausführen, um eine C-Funktion, die einen Wert zurückgibt, passend aufzurufen; anderenfalls kann der Aufruf zwar ausgeführt werden, der zurückgegebene Wert geht aber verloren.

Hinweis BASIC-**DEF FN**-Funktionen und -**GOSUB**-Unterroutinen können nicht aus anderen Sprachen heraus aufgerufen werden.

C.2.2 Voraussetzungen für Benennungsvereinbarungen

Der Ausdruck "Benennungsvereinbarung" (Naming Convention) bezieht sich auf die Art und Weise, in der ein Compiler den Namen einer Routine umändert, bevor er diese in einer Objektdatei plaziert.

Es ist wichtig, daß Sie eine kompatible Benennungsvereinbarung wählen, wenn Sie einen mehrsprachigen Aufruf einführen. Falls der Name der aufgerufenen Routine in jeder Objektdatei unterschiedlich gespeichert wird, ist der Linker nicht in der Lage, eine Entsprechung zu finden und meldet einen nicht gefundenen Verweis.

Microsoft-Compiler plazieren Maschinen-Code in Objektdateien; dort legen sie aber auch die Namen aller Routinen und Variablen ab, auf die ein globaler Zugriff (public) gewährleistet sein muß. Auf diese Weise kann der Linker den Namen einer Routine, die in einem Modul aufgerufen wird, mit dem einer Routine, die in einem anderen Modul definiert ist, vergleichen und eine Übereinstimmung feststellen. Namen werden im ASCII-Format gespeichert. Wie diese gespeichert sind, können Sie genau sehen, wenn Sie das Hilfsprogramm **DEBUG** verwenden, um sich die Bytes einer Objektdatei ausgeben zu lassen.

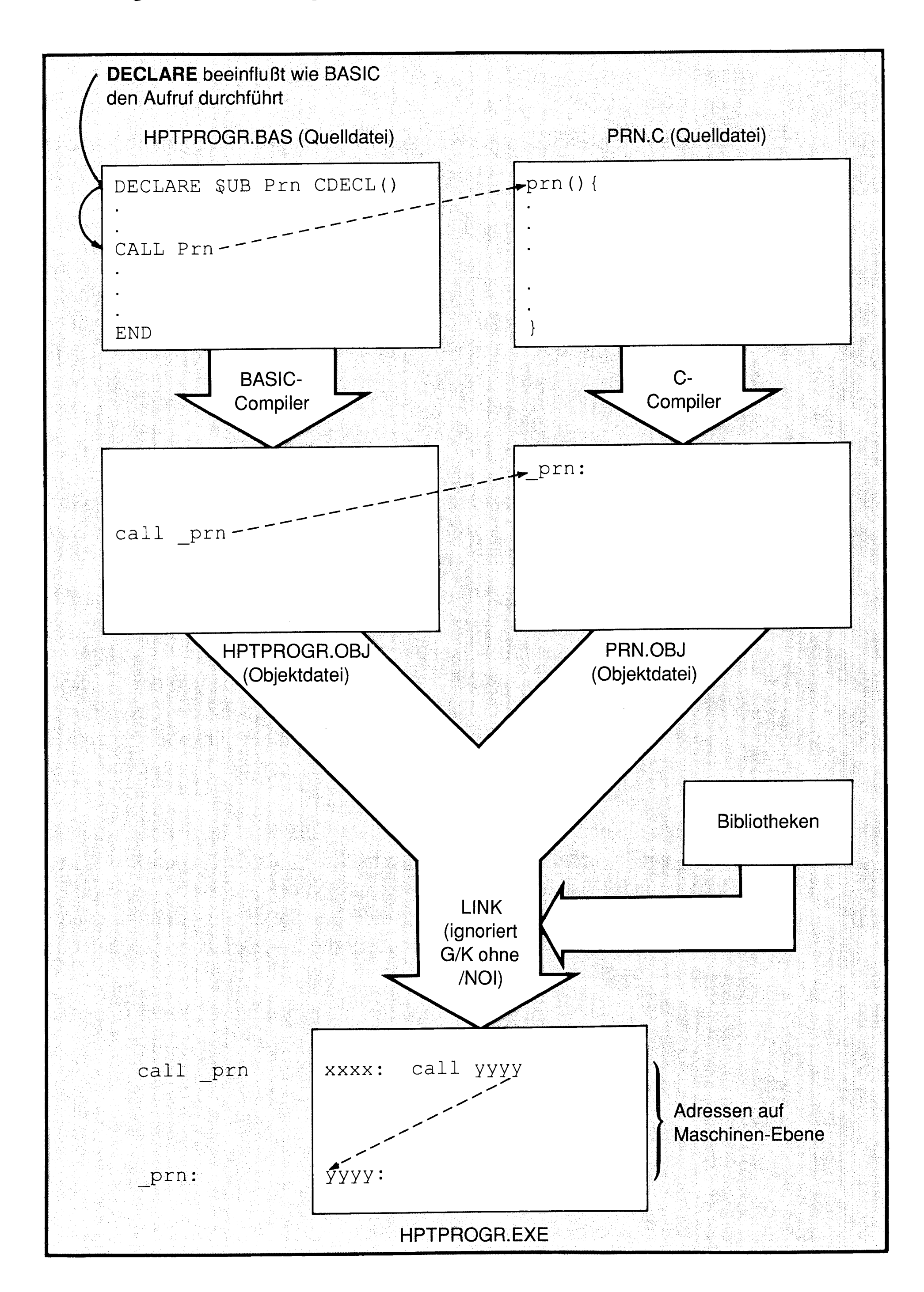
Die Benennungsvereinbarung von BASIC übersetzt alle Buchstaben in Großbuchstaben und verzichtet auf die BASIC-Typdeklarationszeichen (%, &, !, # oder $). BASIC beachtet die ersten 40 Zeichen jedes Namens.

C verwendet eine andere Vereinbarung; der C-Compiler übersetzt nicht alle Buchstaben in Großbuchstaben, fügt aber ein führendes Unterstreichungszeichen (_) vor jedem Routinennamen ein. C beachtet nur die ersten 31 Zeichen eines Namens. Falls ein Name länger ist, werden die zusätzlichen Zeichen einfach nicht in der Objektdatei abgelegt. Auch werden, wenn in der BASIC-Anweisung **DECLARE** das Mehrsprachen-Schlüsselwort **CDECL** angegeben ist, Punkte innerhalb eines Namens von BASIC in Unterstreichungszeichen umgewandelt (zusätzlich zur Hinzufügung des führenden Unterstreichungszeichens).

Solange Sie nicht die Linker-Option **/NOI** (die den Linker dazu veranlaßt, zwischen "Prn" und "prn" zu unterscheiden) verwenden, werden Unterschiede in den Benennungsvereinbarungen automatisch von den Mehrsprachen-Schlüsselwörtern für Sie verwaltet. Bei C-Modulen bedeutet dies, daß Sie darauf achten müssen, sich bei der Benennung Ihrer C-Funktionen nicht auf die Unterschiede zwischen Groß- und Kleinbuchstaben zu verlassen.

Abbildung C.2 erläutert ein komplettes mehrsprachiges Entwicklungsbeispiel und zeigt, wie Benennungsvereinbarungen in den Prozeß eingehen.

Abbildung C.2 Benennungsvereinbarungen



Beachten Sie in dem obigen Beispiel, daß der BASIC-Compiler ein führendes Unterstreichungszeichen vor `Prn` einfügt, wenn der Name in der Objektdatei gespeichert wird. Er tut dies, weil das Schlüsselwort **CDECL** den BASIC-Compiler anweist, die C-Benennungsvereinbarung zu verwenden. Wenn dieses Schlüsselwort verwendet wird, überträgt BASIC außerdem alle Buchstaben in Kleinbuchstaben. (Die Übertragung von Buchstaben in Kleinbuchstaben ist nicht Teil der C-Benennungsvereinbarungen, steht aber in Übereinstimmung zu dem Stil, in dem die meisten C-Programme programmiert sind.)

C.2.3 Voraussetzungen für Aufrufvereinbarungen

Der Begriff "Aufrufvereinbarung" (Calling Convention) bezieht sich auf die Art und Weise, wie eine Sprache einen Aufruf durchführt. Die Wahl der Aufrufvereinbarung betrifft die direkten Maschinenbefehle, die ein Compiler erzeugt, um einen Funktions- oder Prozeduraufruf auszuführen (und aus diesem zurückzukehren).

Die Aufrufvereinbarung ist ein Protokoll auf unterer Ebene. Es ist entscheidend, daß die beiden betroffenen Prozeduren (die Routine, die einen Aufruf durchführt, und die Routine, die aufgerufen wird) dasselbe Protokoll erkennen. Andernfalls kann der Compiler widersprüchliche Anweisungen erhalten, die zu einem unvorhersehbaren Ergebnis führen.

Die Verwendung von Aufrufvereinbarungen wirkt sich auf zwei Arten auf die Programmierung aus:

1. Die aufrufende Routine verwendet eine Aufrufvereinbarung, um festzulegen, in welcher Reihenfolge Argumente (Parameter) an eine andere Routine zu *übergeben* sind. Diese Vereinbarung kann entweder der Standardvereinbarung dieser Sprache entsprechen oder in einer Mehrsprachen-Schnittstelle festgelegt werden. In dem folgenden Beispiel überschreibt das Schlüsselwort **CDECL** in der BASIC-Deklaration für die C-Funktion die BASIC-Standardvereinbarung und veranlaßt, daß die Parameter in der Reihenfolge übergeben werden, in der eine C-Funktion es erwartet, die Parameter zu erhalten:

   ```
   DECLARE Funk1 CDECL (n%, m%)
   ```

2. Die aufgerufene Routine verwendet eine Aufrufvereinbarung, um festzulegen, in welcher Reihenfolge die an sie übergebenen Parameter zu *empfangen* sind. Bei einer C-Funktion kann diese Vereinbarung in der Definition der Funktion angegeben werden.

   In dem folgenden Beispiel überschreibt das Schlüsselwort **fortran** in der Funktionsdefinition die C-Standardvereinbarung und veranlaßt die C-Funktion, die Parameter in Übereinstimmung mit der BASIC-Standardvereinbarung zu empfangen:

```
int fortran funk2 (int x, int y)
            {
             /* Körper der Funktion */
            }
```

Mit anderen Worten, die Art, in der die Funktion in BASIC deklariert ist, veranlaßt den BASIC-*Aufruf* dazu, eine bestimmte Aufrufvereinbarung einzusetzen, während die *Definition* der C-Funktion eine Aufrufvereinbarung angibt oder annimmt. Die beiden Vereinbarungen müssen kompatibel sein. Es ist das Einfachste, die Aufrufvereinbarungen der aufgerufenen Routine zu übernehmen. Zum Beispiel würde eine C-Funktion ihre eigene Vereinbarung für den Aufruf einer anderen C-Funktion verwenden, muß aber die BASIC-Vereinbarungen verwenden, um BASIC aufzurufen. Dies liegt daran, daß BASIC immer seine eigenen Vereinbarungen benutzt, um Parameter zu empfangen. Weil die BASIC- und C-Aufrufvereinbarungen unterschiedlich sind, müssen Sie entweder in der aufrufenden oder in der aufgerufenen Routine eine Mehrsprachen-Schnittstelle angeben, aber nicht in beiden.

C.2.4 Voraussetzungen für die Parameterübergabe

Der Abschnitt C.2.3 erläuterte das umfassende Protokoll (Aufrufvereinbarung), das zwei Routinen verwenden, um Daten miteinander auszutauschen; dieser Abschnitt erläutert die Art, wie ein einzelnes Datum (ein Parameter) tatsächlich gesendet wird.

Wenn Ihre Routinen nicht darin übereinstimmen, wie ein Parameter zu senden ist, dann wird die aufgerufene Routine falsche Daten empfangen. Darüber hinaus ist es möglich, daß das Programm das System dazu veranlassen könnte, unvorhersehbare Ergebnisse zu liefern.

Microsoft-Compiler stellen für die Übergabe eines Parameters drei Methoden zur Verfügung:

Methode	***Beschreibung***
Mit kurzer (near) Referenz	Übergibt die kurze Adresse (Offset) einer Variablen. Diese Methode ermöglicht der aufgerufenen Routine direkten Zugriff auf die Variable selbst. Jede Änderung, die die Routine an dem Parameter vornimmt, wird in die aufrufende Routine zurückgegeben.

Methode	*Beschreibung*
Mit langer Referenz (far)	Übergibt die lange Adresse (Intersegment) einer Variablen. Abgesehen davon, daß eine längere Adresse übergeben wird, ist diese Methode gleich zur Übergabe mit kurzer Referenz.
Als Wert	Übergibt nur den Wert einer Variablen, nicht deren Adresse. Bei dieser Methode kennt die aufgerufene Routine zwar den Wert des Parameters, hat aber keinen Zugriff auf die eigentliche Variable. Sobald die Routine beendet ist, haben Änderungen eines Werteparameters keine Auswirkung mehr auf den Wert des Parameters in der aufrufenden Routine.

Die Tatsache, daß es unterschiedliche Methoden zur Parameterübergabe gibt, hat zwei Auswirkungen auf die mehrsprachige Programmierung:

1. Sie müssen sicherstellen, daß die aufrufende Routine sowie die aufgerufene Routine für die Parameterübergabe dieselbe Methode verwenden. Normalerweise müssen Sie die Standard-Parameterübergabe, die von jeder Sprache verwendet wird, prüfen und, falls notwendig, Anpassungen vornehmen. Jede Sprache kennt Schlüsselwörter oder Spracheigenschaften, die es Ihnen ermöglichen, die Methoden bei der Parameterübergabe zu verändern.
2. Die Programmlogik kann es erfordern, daß Sie eine spezielle Methode der Parameterübergabe einsetzen, anstatt die Standards jeder Sprache zu benutzen.

Die folgende Tabelle faßt die Standards der Parameterübergabe für BASIC und C zusammen.

Sprache	*Kurze Referenz*	*Lange Referenz*	*Als Wert*
BASIC	Alle	---	---
C	Kurze Datenfelder	Lange Datenfelder	Nicht-Datenfelder

Informationen zum Überschreiben der Standards finden Sie in Abschnitt C.3, "BASIC-Aufrufe zu C", und Abschnitt C.4, "C-Aufrufe zu BASIC".

C.2.5 Kompilieren und Binden

Nachdem Sie Ihre Quelldatei geschrieben haben und die in den Abschnitten C.2.2 bis C.2.4 erörterten Probleme gelöst haben, sind Sie in der Lage, einzelne Module zu kompilieren und diese anschließend zu binden.

C.2.5.1 Wie Sie mit dem geeigneten Speichermodell kompilieren

Mit BASIC sind keine besonderen Optionen erforderlich, um Quelldateien zu kompilieren, die Teil eines mehrsprachigen Programmes sind.

Mit Microsoft C sind dagegen nicht alle Speichermodelle kompatibel zu anderen Sprachen. BASIC verwendet nur lange (Intersegment) Code-Adressen. Daher müssen Sie C-Module immer mit Medium-, Large- oder Huge-Speichermodellen kompilieren, weil auch diese Modelle lange Code-Adressen verwenden. Das Kompilieren mit einem Small- oder Compact-Speichermodell führt beim mehrsprachigen Programm zu unvorhersehbaren Ergebnissen, sobald ein Aufruf zu oder aus C erfolgt. (Dieses Problem kann vermieden werden, wenn Sie in einem Programm, das mit einem Small- oder Compact-Speichermodell arbeitet, in der Definition einer C-Funktion das Schlüsselwort **far** verwenden. Dies legt fest, daß die Funktion einen langen Aufruf bzw. Rücksprung benutzt.)

Hinweis Microsoft QuickC verwendet das Medium-Speichermodell, wenn Sie den Befehl "Kompilieren" aus dem Menü "Ausführen" und "Obj" aus den "Ausgabe-Optionen" verwenden. Wenn Sie jedoch von der Befehlszeile aus entweder mit dem Befehl **QCL** oder dem Befehl **CL** kompilieren, müssen Sie das richtige Speichermodell angeben; andernfalls wird das Small-Modell verwendet, und Ihre C-Objektdateien werden nicht kompatibel zu Ihren BASIC-Objektdateien sein.

Der vorhergehende Absatz betrifft die Größe von Code-Adressen. Unterschiede in der Größe von Daten-Adressen können mit Compiler-Optionen oder im Quellcode aufgehoben werden. Die Wahl des Speichermodells hat Auswirkungen auf die Größe des Standard-Datenzeigers in C, obwohl dieser Standard mit **near** und **far** überschrieben werden kann. Die Wahl des Speichermodells hat in C ebenso Auswirkungen darauf, ob Daten-Objekte im Standard-Datensegment abgelegt werden. Falls ein Daten-Objekt nicht im Standard-Datensegment abgelegt ist, kann es nicht direkt mit kurzer Referenz übergeben werden.

C.2.5.2 Wie Sie mit Sprach-Bibliotheken binden

In vielen Fällen kann das Binden von Modulen, die mit unterschiedlichen Sprachen kompiliert sind, einfach durchgeführt werden. Jeder der folgenden Schritte stellt sicher, daß alle erforderlichen Bibliotheken in der richtigen Reihenfolge gebunden werden:

- Legen Sie alle Sprach-Bibliotheken in demselben Verzeichnis ab, in dem sich auch die Quelldateien befinden.
- Führen Sie Verzeichnisse, die alle benötigten Bibliotheken enthalten, in der Umgebungsvariablen **LIB** auf.
- Veranlassen Sie den Linker dazu, Anfragen nach Bibliotheken auszugeben.

Unter der Annahme, daß das BASIC-Modul auf der **link**-Befehlszeile zuerst angegeben ist (und die Linker-Option **/NOD** *nicht* angegeben ist), findet der Linker für jeden der oben aufgeführten Fälle Bibliotheken in der Reihenfolge, in der er diese benötigt. Falls Sie die Bibliotheken auf der **link**-Befehlszeile eingeben, müssen die BASIC-Bibliotheken vor allen anderen stehen, weil BASIC es erfordert, daß seine Bibliotheken vor solchen einer anderen Sprache durchsucht werden.

Hinweis Wenn Sie BASIC-Module mit solchen anderer Sprachen binden, muß das erste angegebene Modul das BASIC-Hauptmodul sein. Dies legt darüber hinaus die Standard-Reihenfolge fest, in der Bibliotheken durchsucht werden.

C.3 BASIC-Aufrufe zu C

Microsoft BASIC unterstützt Aufrufe zu Routinen, die in Microsoft C geschrieben sind. Der Abschnitt C.3 beschreibt die für C-Aufrufe notwendige Syntax. Um die Darstellung von Konzepten einfach zu halten, werden in diesem Beispiel nur Ganzzahlen als Parameter verwendet.

Der Abschnitt C.4 beschreibt Einschränkungen bei der Verwendung von Funktionen aus der C-Standard-Bibliothek. Schlagen Sie in diesem Abschnitt nach, falls Sie irgendeine System-Bibliotheksfunktion oder eine Bibliotheksfunktion zur Speicherzuweisung verwenden.

Informationen darüber, wie bestimmte Arten von Daten übergeben werden müssen, finden Sie in den Abschnitten C.5 bis C.8.

C.3.1 Die BASIC-Schnittstelle zu anderen Sprachen

Die BASIC-Anweisung **DECLARE** bietet eine flexible und komfortable Schnittstelle zu anderen Sprachen. Sie ist verfügbar mit Microsoft QuickBASIC Version 4.0 und größer. Frühere Versionen von BASIC, die die Anweisung **DECLARE** nicht unterstützen, unterstützen ebenfalls keine Bibliotheken, die kompatibel mit anderen Sprachen sind.

Weitere Informationen zur Syntax der Anweisung **DECLARE** finden Sie in Abschnitt C.3.1.1, "Die Anweisung DECLARE", in diesem Handbuch und in den Erläuterungen der **DECLARE**-Anweisung im *BASIC-Befehlsverzeichnis* und in *Programmieren in BASIC: Ausgewählte Themen.*

C.3.1.1 Die Anweisung DECLARE

Die Syntax der Anweisung **DECLARE** unterscheidet sich leicht für **FUNCTION**- bzw. **SUB**-Prozeduren. Für **FUNCTION**-Prozeduren lautet die Syntax der **DECLARE**-Anweisung wie folgt:

DECLARE FUNCTION *Name*[**CDECL**][**ALIAS** *"Aliasname"*][([*Parameterliste*])]

Für **SUB**-Prozeduren lautet die Syntax der **DECLARE**-Anweisung:

DECLARE SUB *Name*[**CDECL**][**ALIAS** *"Aliasname"*][([*Parameterliste*])]

Das Feld *Name* ist der Name, der in der BASIC-Quelldatei für die **SUB**- oder **FUNCTION**-Prozedur, die Sie aufrufen möchten, erscheint. Es folgen die Schritte, die empfohlen werden, um die Anweisung **DECLARE** für Aufrufe anderer Sprachen einzusetzen:

1. Für jede einzelne zwischensprachliche Routine, die Sie aufzurufen beabsichtigen, setzen Sie eine **DECLARE**-Anweisung an den Beginn eines jeden Moduls, in dem die Routine aufgerufen wird. (QuickBASIC kann die **DECLARE**-Anweisungen für anderssprachige Routinen nicht automatisch erzeugen.)

   Zum Beispiel könnte Ihr Programm das Unterprogramm `Maxparam` fünfmal aufrufen, jedesmal mit unterschiedlichen Argumenten. Sie müssen `Maxparam` jedoch nur einmal für jedes Modul deklarieren. Die **DECLARE**-Anweisungen müssen ziemlich am Anfang des Modules stehen und allen ausführbaren Anweisungen vorangehen. Eine gute Möglichkeit, dies zu erreichen, besteht in der Verwendung einer Include-Datei.

2. Falls Sie eine Routine aufrufen, die in einem C-Modul definiert ist, verwenden Sie in der **DECLARE**-Anweisung **CDECL** (es sei denn, die C-Routine ist mit dem Schlüsselwort **pascal** oder **fortran** definiert).

   CDECL weist BASIC an, während jedes folgenden Aufrufs von *Name* die C-Benennungs- und Aufrufvereinbarungen zu verwenden.

3. Falls Sie eine C-Funktion mit einem Namen aufrufen, der Zeichen enthält, die in BASIC unzulässig sind (zum Beispiel das Unterstreichungszeichen), können Sie das Problem mit **ALIAS** ungehen. Eine Erklärung von **ALIAS** finden Sie in Abschnitt C.3.1.2. Wenn Sie das Schlüsselwort **CDECL** verwenden, können Sie anstelle des Unterstreichungszeichens einen Punkt einsetzen. BASIC ersetzt den Punkt dann durch ein Unterstreichungszeichen.

4. Verwenden Sie die Parameterliste, um zu bestimmen, wie jeder Parameter übergeben werden muß. Weitere Informationen zur Verwendung einer Parameterliste finden Sie in Abschnitt C.3.1.3.

5. Nachdem die Routine einmal richtig deklariert ist, rufen Sie diese genauso auf, wie Sie es mit einem BASIC-Unterprogramm oder einer BASIC-Funktion tun würden.

Die anderen Felder sind in den folgenden Ausführungen erläutert.

C.3.1.2 Wie Sie ALIAS verwenden

Wie oben erwähnt, kann die Verwendung des Schlüsselwörters **ALIAS** notwendig werden, wenn Sie ein Unterstreichungszeichen als Teil des C-Bezeichners einsetzen möchten. Außerdem schreibt C, obwohl dies wahrscheinlich kein Problem sein wird, weniger Zeichen eines Namens in eine Objektdatei (31, zusätzlich zu dem führenden Unterstreichungszeichen) als BASIC, das bis zu 40 Zeichen eines Namens in eine Objektdatei schreibt.

Hinweis Sie benötigen das Schlüsselwort **ALIAS** *nicht*, um die Typdeklarationszeichen (%, &, !, #, $) zu entfernen. BASIC entfernt diese Zeichen automatisch, wenn es Objektcode erzeugt. Daher stimmt `Fakt%` in BASIC mit `fakt` in C überein.

Das Schlüsselwort **ALIAS** weist BASIC an, *Aliasname* anstatt *Name* in die Objektdatei zu schreiben. Die BASIC-Quelldatei enthält immer noch Aufrufe für *Name*. Diese Aufrufe werden jedoch so interpretiert, als wären Sie eigentlich Aufrufe für *Aliasname*.

Beispiel

In dem unteren Beispiel schreibt BASIC den *Aliasnamen* `quad_ergeb` und nicht den Namen `QuadErgeb` in den Objektcode. Dies erspart es der C-Funktion ein Kennzeichen für gemischte Schreibweise zu verwenden, bietet aber denselben Grad an Wiedererkennbarkeit wie der BASIC-Name.

```
DECLARE FUNCTION QuadErgeb% ALIAS "quad_ergeb" (a, b, c)
```

C.3.1.3 Wie Sie die Parameterliste einsetzen

Die Syntax für *Parameterliste*, gefolgt von Erklärungen zu jedem Feld, sehen Sie weiter unten. Beachten Sie, daß Sie **BYVAL** oder **SEG** verwenden können, aber nicht beide.

Syntax

[{**BYVAL** | **SEG**}] *Variable* [**AS** *Typ*] [,[{**BYVAL** | **SEG**}] *Variable* [**AS** *Typ*]]...

Verwenden Sie das Schlüsselwort **BYVAL**, um einen Werteparameter zu deklarieren. In jedem nachfolgenden Aufruf wird das entsprechende Argument als Wert übergeben (die Standardmethode für C-Module).

Hinweis BASIC bietet zwei Möglichkeiten für "Übergabe als Wert". Die übliche Methode für Übergabe als Wert besteht darin, ein zusätzliches Klammernpaar zu verwenden, wie in

```
CALL Preis((A))
```

Diese Methode erzeugt eigentlich einen temporären Wert, dessen Adresse übergeben wird. **BYVAL** bietet eine echte Methode zur Übergabe als Wert, da der Wert selbst übergeben wird, nicht eine Adresse. Nur wenn **BYVAL** verwendet wird, ist ein BASIC-Programm kompatibel zu einer Nicht-BASIC-Routine, die einen Werteparameter erwartet.

Verwenden Sie das Schlüsselwort **SEG**, um einen Parameter mit langer Referenz zu deklarieren. In jedem nachfolgenden Aufruf wird die lange (Intersegment) Adresse des entsprechenden Argumentes übergeben. Informationen und Warnungen für die Verwendung des Schlüsselwörters **SEG** finden Sie in der Beschreibung der Anweisung **DECLARE** sowie in Kapitel 2, "Datentypen", im *BASIC-Befehlsverzeichnis.*

Für *Variable* können Sie zwar jeden zulässigen Namen wählen, aber nur der mit dem Namen verknüpfte Typ hat überhaupt Bedeutung für BASIC. Genauso wie mit anderen Variablen kann der Typ mit einem Typdeklarationszeichen (%, &, !, #, $) oder durch implizite Deklaration angezeigt werden.

Die Klausel **AS** *Typ* überschreibt die Standard-Typdeklaration von *Variable.* Das Feld *Typ* kann **INTEGER, LONG, SINGLE, DOUBLE, STRING** oder ein benutzerdefinierter Typ sein. Oder es kann den Typ **ANY** haben, der BASIC anweist, jeden Datentyp zuzulassen, der als Argument übergeben werden soll.

Beispiele

In dem folgenden Beispiel ist `Kalk2` als C-Routine deklariert, die drei Argumente hat: Die ersten beiden sind Ganzzahlen, die als Wert übergeben werden, und das letzte ist eine reelle Zahl einfacher Genauigkeit, die als Wert übergeben wird.

```
DECLARE FUNCTION Kalk2! CDECL (BYVAL a%, BYVAL b%, BYVAL c!)
```

Das nächste Beispiel deklariert das Unterprogramm `Maxaus`, das als Argumente eine mit langer Referenz übergebene Ganzzahl und eine als Wert übergebene reelle Zahl doppelter Genauigkeit hat.

```
DECLARE SUB Maxaus (SEG var1 AS INTEGER, BYVAL var2 AS
DOUBLE)
```

C.3.2 Alternative BASIC-Schnittstellen

Obwohl die **DECLARE**-Anweisung eine besonders bequeme Schnittstelle bietet, gibt es weitere Methoden zur Durchführung mehrsprachiger Aufrufe.

Anstatt das Verhalten von BASIC mit **CDECL** zu verändern, können Sie das Verhalten von C verändern, indem Sie das Schlüsselwort **pascal** oder **fortran** in die Definition der Funktion einfügen. (Diese beiden Schlüsselwörter sind in ihrer Funktion identisch.) Sie können das C-Modul aber auch mit der Option **/Gc** kompilieren, die festlegt, daß alle C-Funktionen, Aufrufe und globale Symbole die Vereinbarungen von BASIC/FORTRAN/Pascal verwenden.

Zum Beispiel verwendet die folgende C-Funktion die BASIC/FORTRAN/Pascal-Vereinbarungen, um einen Ganzzahl-Parameter zu empfangen:

```
int pascal fun1 (n)
int n;
{
 }
```

Sie können Methoden zur Parameterübergabe wie folgt festlegen, selbst wenn Sie die **DECLARE**-Anweisung oder die Parameterliste oder beide auslassen:

- Sie können den Aufruf mit der Anweisung **CALLS** ausführen. Die Anweisung **CALLS** veranlaßt, daß jeder Parameter mit langer Referenz übergeben wird.
- In der tatsächlichen Parameterliste können Sie die Schlüsselwörter **BYVAL** und SEG verwenden, wenn Sie den Aufruf ausführen.

Im folgenden Beispiel haben **BYVAL** und **SEG** dieselbe Bedeutung, die sie auch in einer BASIC-Anweisung **DECLARE** haben. Wenn Sie **BYVAL** und **SEG** auf diese Weise einsetzen, müssen Sie jedoch sehr vorsichtig sein, weil weder Typ noch Anzahl der Parameter geprüft werden (wie dies der Fall wäre, wenn eine **DECLARE**-Anweisung eingesetzt würde). Beachten Sie auch, daß Sie entweder das Schlüsselwort **fortran** oder das Schlüsselwort **pascal** in der Definition der C-Funktion verwenden oder die C-Funktion mit der Option **/Gc** kompilieren müssen, falls Sie keine **DECLARE**-Anweisung verwenden.

```
CALL Fun2 (BYVAL Term1, BYVAL Term2, SEG Sum);
```

C.3.3 BASIC-Aufrufe zu C

Dieser Abschnitt wendet die in Abschnitt C.3.1.1 aufgeführten Schritte in zwei Beispielprogrammen an. Eine Analyse der Aspekte für das Programmieren folgt jedem Beispiel.

Hinweis QuickBASIC bietet eine BASIC-Systemebenen-Funktion (**B_OnExit**), die von anderssprachigen Routinen aufgerufen werden kann, um eine Beendigungsprozedur anzumelden, die aufgerufen wird, wenn ein BASIC-Programm endet oder erneut gestartet wird, sofern eine Quick-Bibliothek vorhanden ist. Die Beschreibung dieser Routine finden Sie im Abschnitt C.9.

C aus BASIC ohne Rückgabewert aufrufen

Das folgende Beispiel erläutert ein BASIC-Hauptmodul, das eine C-Funktion, `maxparam`, aufruft. Die Funktion `maxparam` gibt keinen Wert zurück, paßt aber das kleinere der beiden Argumente so an, daß es dem größeren Argument gleich wird.

Beispiel

```
' BASIC-Quelldatei - ruft C-Funktion auf, die keinen
' Wert zurückgibt
'
' DECLARE SUB Maxparam CDECL (A AS INTEGER, B AS INTEGER)
'
' Als Unterprogramm deklariert, da kein Rückgabewert
' vorhanden ist. Das Schlüsselwort CDECL bewirkt, daß
' der Aufruf von Maxparam mit den C-Vereinbarungen
' durchgeführt wird. Die Ganzzahl-Parameter werden mit
' kurzer Referenz übergeben (BASIC-Standard).
'
X% = 5
Y% = 7
PRINT USING "X% = ##  Y% = ##";X% ;Y% ' X% und Y%
                                      ' vor dem Aufruf
CALL Maxparam (X%, Y%)                ' Aufruf der C-
                                      ' Funktion
PRINT USING "X% = ##  Y% = ##";X% ;Y% ' X% und Y% nach
                                      ' dem Aufruf
END
/* C-Quelldatei */
/* Kompilieren mit dem MEDIUM- oder LARGE-Speichermodell */
/* Maxparam wird mit VOID deklariert, weil es keinen
Rückgabewert gibt */
void maxparam (p1, p2)
int near *p1;   /* Ganzzahl-Parameter werden mit kurzer
Referenz empfangen */
int near *p2;   /* Schlüsselwort NEAR wird mit dem MEDIUM-
Modell nicht benötigt. */
```

```
{
        if (*p1 > *p2)
              *p2 = *p1;
        else
              *p1 = *p2;
}
```

Sie sollten die folgenden Programmieraspekte im Gedächtnis behalten, wenn Sie C aus BASIC heraus ohne Rückgabewerte aufrufen:

- **Benennungsvereinbarungen**

  Das Schlüsselwort **CDECL** bewirkt, daß `Maxparam` mit den Benennungsvereinbarungen von C aufgerufen wird (als `_maxparam`).

- **Aufrufvereinbarungen**

  Das Schlüsselwort **CDECL** bewirkt, daß `Maxparam` mit den Aufrufvereinbarungen von C aufgerufen wird, mit denen Parameter in umgekehrter Reihenfolge zu ihrem Erscheinen im Quellcode übergeben werden.

- **Methoden zur Parameterübergabe**

  Da die C-Funktion `Maxparam` den Wert eines der Parameter ändern könnte, müssen beide Parameter als Referenz übergeben werden. In diesem Fall wurde kurze Referenz gewählt; diese Methode ist Standard für BASIC (so daß weder **BYVAL** noch **SEG** verwendet werden) und ist in C festgelegt, indem kurze Zeiger (Near Pointer) verwendet werden.

  Lange Referenz hätte angegeben werden können, indem jedem Argument in der **DECLARE**-Anweisung **SEG** zugefügt worden wäre. In einem solchen Fall würden die C-Parameter-Deklarationen lange Zeiger (Far Pointer) verwenden.

C aus BASIC mit einem Funktionsaufruf aufrufen

Das folgende Beispiel demonstriert ein BASIC-Hauptmodul, das eine C-Funktion, `Fakult`, aufruft. Diese Funktion gibt die Fakultät eines ganzzahligen Wertes zurück.

Beispiel

```
' BASIC-Quelldatei - ruft C-Funktion mit
' Rückgabewert auf
'
DECLARE FUNCTION Fakult% CDECL (BYVAL N AS INTEGER)
'
```

```
' Als Funktion deklariert, die eine Ganzzahl (%)
' zurückgibt
' Das Schlüsselwort CDECL bewirkt, daß der Aufruf
' zu Fakult% mit C-Vereinbarungen durchgeführt wird.
' Ganzzahlparameter wird als Wert übergeben.
'

X% = 3
Y% = 4
PRINT USING "Die Fakultät von X%    ist ####"; Fakult%(X%)
PRINT USING "Die Fakultät von Y%    ist ####"; Fakult%(Y%)
PRINT USING "Die Fakultät von X%+Y% ist ####";
Fakult%(X%+Y%)
END
/* C-Quelldatei */
/* Kompiliert mit dem MEDIUM- oder LARGE-Modell */
/* Funktion für Fakultät, gibt Ganzzahl zurück */
int fakult(n)
int n;        /* Ganzzahl übergeben als Wert,*/
              /* C-Standard */
{
       int    ergebn = 1;
       while (n > 0)
                 ergebn *= n--;
                 /* Parameter n wird hier verändert */
       return(ergebn);
}
```

Sie sollten die folgenden Programmieraspekte im Gedächtnis behalten, wenn Sie C aus BASIC heraus mit einem Funktionsaufruf aufrufen:

- **Benennungsvereinbarungen**

  Das Schlüsselwort **CDECL** bewirkt, daß `Fakult` mit den C-Benennungsvereinbarungen aufgerufen wird (als `_fakult`).

- **Aufrufvereinbarungen**

  Das Schlüsselwort **CDECL** bewirkt, daß `fakult` mit der Aufrufvereinbarung von C aufgerufen wird, mit der Parameter in umgekehrter Reihenfolge übergeben werden, und die andere kleine Unterschiede festlegt.

- **Methoden zur Parameterübergabe**

  Die obige C-Funktion sollte den Parameter als Wert empfangen. Andernfalls wird die Funktion den Wert des Parameters in dem aufrufenden Modul verfälschen. Echte Übergabe als Wert wird in BASIC nur ausgeführt, wenn in der **DECLARE**-Anweisung des Parameters **BYVAL** verwendet wird; in C ist Übergabe als Wert der Standard (außer für Datenfelder).

C.3.4 Einschränkungen für Aufrufe aus BASIC

BASIC besitzt eine weitaus komplexere Umgebung sowie Initialisierungs-Prozedur als C. Zwischensprachlicher Aufruf zwischen BASIC und anderen Sprachen ist nur möglich, weil BASIC eine Reihe von Aufrufen für Bibliotheksfunktionen aus der anderen Sprache abfängt und diese Aufrufe auf seine eigene Weise behandelt. Mit anderen Worten, BASIC erstellt eine geeignete Umgebung, in der die C-Routinen funktionieren können.

BASIC ist jedoch in seiner Fähigkeit, einige C-Funktionsaufrufe zu behandeln, begrenzt. In diesem Abschnitt werden zwei Arten der Begrenzung dargestellt: C-Speicherzuweisungs-Funktionen, die evtl. eine besondere Deklaration erfordern, und einige spezielle C-Bibliotheksfunktionen, die überhaupt nicht aufgerufen werden können.

C.3.4.1 Speicherzuweisung

Falls Ihr C-Modul ein Medium-Modell benutzt, und Sie dynamische Speicherzuweisungen mit **malloc( )** vornehmen, oder falls Sie ausdrückliche Aufrufe von **_nmalloc( )** mit einem beliebigen Speichermodell ausführen, dann müssen Sie, bevor Sie C aufrufen, die folgenden Zeilen in Ihren BASIC-Quellcode einfügen:

```
DIM mallocbuf% (2048)
COMMON SHARED /NMALLOC/ mallocbuf%()
```

Das Datenfeld darf jeden beliebigen Namen haben; nur die Größe des Datenfeldes ist von Bedeutung. Der Name des Common-Blockes muß jedoch **NMALLOC** sein. In der QuickBASIC 4.0-Speicherumgebung müssen Sie diese Deklaration in ein Modul schreiben, das Sie in eine Quick-Bibliothek einfügen. Weitere Informationen zu Quick-Bibliotheken finden Sie in Kapitel 8, "Quick-Bibliotheken", und Kapitel 9, "Wie Sie aus DOS heraus kompilieren und binden".

Das obere Beispiel reserviert 4K Speicherplatz in dem Common-Block **NMALLOC**. Wenn BASIC C-**malloc**-Aufrufe abfängt, weist BASIC Speicherplatz aus diesem Common-Block zu.

Warnung Wenn Sie die eingebaute BASIC-Funktion **CLEAR** aufrufen, geht jeglicher Speicherplatz, der mit near-**malloc**-Aufrufen zugewiesen wurde, verloren. Sofern Sie überhaupt **CLEAR** verwenden, dann nur vor Aufrufen von **malloc**.

Wenn Sie in mehrsprachigen Programmen lange Speicheranforderungen (Far Memory Request) ausführen, werden Sie es wahrscheinlich hilfreich finden, zunächst die eingebaute BASIC-Funktion **SETMEM** aufzurufen. Diese Funktion kann dazu eingesetzt werden, den von BASIC beanspruchten Speicherplatz zu reduzieren und somit Speicher für lange Zuweisungen freizugeben.

C.3.4.2 Inkompatible Funktionen

Die folgenden C-Funktionen sind inkompatibel zu BASIC und sollten vermieden werden:

- Alle Formen von **spawn()** und **exec()**
- **system()**
- **getenv()**
- **putenv()**

Ein Aufruf dieser Funktionen führt zu der BASIC-Fehlermeldung

```
Erweiterte Eigenschaft nicht verfügbar.
```

Zusätzlich sollten Sie nicht mit den *x***VARSTK.OBJ**-Modulen (wobei *x* ein Speichermodell ist) binden, die C bietet, um Speicher vom Stapel zuzuweisen.

Hinweis Die globalen C-Laufzeitvariablen **environ** und **_pgmptr** werden als NULL definiert. Alle Funktionalitäten dieser Variablen sowie die oben angeführten Funktionen können mit BASIC-Anweisungen und eingebauten Funktionen emuliert werden.

C.3.4.3 Zeichenkettenraum zuweisen

Anderssprachige Routinen können dynamischen Zeichenkettenraum zuweisen, indem sie die folgende BASIC-**FUNCTION** aufrufen:

```
FUNCTION HoleRaum$  (x) STATIC
HoleRaum$ = STRING$ (x, CHR$(0))
END FUNCTION
```

Die Prozedur `HoleRaum` gibt einen kurzen Zeiger auf einen Zeichenkettenbeschreiber zurück, der auf x Bytes des Zeichenkettenraumes zeigt. Weil dieser Raum von BASIC verwaltet wird, kann er sich jedesmal, wenn BASIC-Code ausgeführt wird, bewegen. Daher muß auf diesen Raum mit dem Zeichenkettenbeschreiber zugegriffen werden, und der Zeichenkettenbeschreiber darf nicht durch anderssprachigen Code verändert werden. Um diesen Raum freizugeben, übergeben Sie den kurzen Zeiger auf den Zeichenkettenbeschreiber an die folgende BASIC-**SUB**-Prozedur:

```
SUB FreiRaum (a$) STATIC
a$ = ""
END SUB
```

Um eine Zeichenkette zurückzugeben, die das Ergebnis einer anderssprachigen Routine darstellt, geben Sie einen kurzen Zeiger auf einen statischen Zeichenkettenbeschreiber zurück, der in dem anderssprachigen Code deklariert ist. Da BASIC solche Zeichenketten hin- und herbewegt, wird der statische Zeichenkettenbeschreiber, der von dem anderssprachigen Code zugewiesen wurde, ungültig, sobald die Funktion abgearbeitet ist (oder einen Aufruf zu einer beliebigen BASIC-Prozedur durchführt).

C.3.4.4 E/A mit BASIC-Dateien ausführen

Anderssprachige Routinen können, indem sie BASIC-Prozeduren aufrufen, Eingabe und Ausgabe mit Dateien durchführen, die von der BASIC-Anweisung **OPEN** geöffnet wurden. Das folgende Beispiel ist eine BASIC-**SUB**, die aufgerufen werden kann, um eine Ganzzahl in eine als `dateinr` geöffnete BASIC-Datei zu schreiben.

```
SUB GibAus (dateinr%, x%) STATIC
PRINT #dateinr%, x%
END SUB
```

C.3.4.5 Ereignisse und Fehler

BASIC-Ereignisse, einschließlich **COM**, Tastatur, **TIMER** und **PLAY** können während der Ausführung anderssprachigen Codes auftreten. Der anderssprachige Code kann es erlauben, solche Ereignisse wie folgt zu behandeln:

- Wenn Sie mit dem Befehl **BC** kompilieren, sollten Sie die BASIC-Prozedur mit gesetzter Option Ereignisbehandlung (zum Beispiel **/v** oder **/w**) kompilieren.
- Innerhalb der QuickBASIC-Umgebung geben Sie Ereignisbehandlungs-Syntax einfach in der Prozedur selbst an und verwenden anschließend den Befehl **EXE-Datei erstellen** oder **Bibliothek erstellen** aus dem Menü **Ausführen**, um eine ausführbare Datei zu erzeugen, oder die Prozedur in eine Quick-Bibliothek einzubinden.

Die folgende BASIC-**SUB** ermöglicht es Ihnen, BASIC-Fehler zu deklarieren:

```
SUB MacheFehl (x%) STATIC
ERROR x%
END SUB
```

Wenn Ihre anderssprachige Routine die Fehlernummer an diese Prozedur übergibt, wird die **ERROR**-Anweisung ausgeführt, und BASIC stellt den Stapel bis zu dem vorhergehenden Aufruf zu Nicht-BASIC-Code wieder her. Die BASIC-Anweisung, die den Aufruf zu dem Nicht-BASIC-Code enthält, ist diejenige Anweisung, die **RESUME** erneut ausführt.

C.4 C-Aufrufe zu BASIC

Microsoft C kann in Microsoft BASIC geschriebene Routinen aufrufen, falls das Hauptprogramm in BASIC vorliegt. Die Abschnitte C.4.1 bis C.4.2 beschreiben die notwendige Syntax zum Aufruf von BASIC aus C heraus. Es werden nur einfache Parameterlisten verwendet.

Informationen, wie spezielle Arten von Daten übergeben werden, finden Sie in den Abschnitten C.5 bis C.8.

C.4.1 Die C-Schnittstelle zu anderen Sprachen

Die C-Schnittstelle zu anderen Sprachen verwendet die Standard-C-Anweisung **extern** zusammen mit dem speziellen Schlüsselwort **fortran** oder **pascal**. Die Verwendung einer dieser beiden Schlüsselwörter veranlaßt, daß die Routine mit den Benennungs- und Aufrufvereinbarungen von FORTRAN/Pascal/BASIC aufgerufen wird. Es folgen die empfohlenen Schritte, um mit dieser Anweisung einen Mehrsprachen-Aufruf aus C heraus durchzuführen:

1. Schreiben Sie eine **extern**-Deklaration für jede aufgerufene mehrsprachige Routine.

   Die Anweisung **extern** sollte allen Aufrufen dieser Routine vorhergehen. Die genauen Syntaxregeln zur Verwendung der Schlüsselwörter **fortran** und **pascal** mit der Anweisung **extern** sind weiter unten beschrieben. Anstatt das Schlüsselwort **fortran** oder **pascal** zu verwenden, können Sie einfach mit /**Gc** kompilieren. Die Option /**Gc** veranlaßt, daß für alle Funktionen in dem Modul die BASIC/FORTRAN/Pascal-Benennungs- und Aufrufvereinbarungen verwendet werden. Wenn Sie möchten, daß die meisten (aber nicht alle) der Routinen des Moduls die BASIC/FORTRAN/Pascal-Vereinbarungen verwenden, können Sie das Modul mit **Gc** kompilieren, müssen aber das Schlüsselwort **cdecl** in den Definitionen solcher Funktionen verwenden, für die Sie die Auswirkungen des Schalters **Gc** überschreiben möchten. Bitte beachten Sie bei der Verwendung von **cdecl** in einer C-Funktion, daß es kleingeschrieben werden muß.

2. Verwenden Sie Typdeklarationen für Parameter innerhalb der **extern**-Anweisung.

   Dieser Schritt ist wichtig, falls Sie Zeiger angeben möchten, die nicht die Standardlänge haben. (Kurze Zeiger sind Standard für das Medium-Modell; lange Zeiger sind Standard für das Large-Modell.)

3. Um ein Argument als Referenz zu übergeben, übergeben Sie einen Zeiger auf das Objekt.

   C übersetzt Datenfeldnamen automatisch in Adressen. Daher werden Datenfelder als Referenz übergeben.

4. Nachdem eine Routine einmal richtig mit einer **extern**-Anweisung deklariert ist, rufen Sie diese genauso auf, als würden Sie eine C-Funktion aufrufen.
5. Kompilieren Sie das C-Modul immer mit dem Medium- oder Large-Modell.

Wie die Schlüsselwörter fortran und pascal verwendet werden:

Es gibt zwei Syntaxregeln, die bei der Verwendung der Schlüsselwörter **fortran** oder **pascal** anzuwenden sind:

1. Die Schlüsselwörter **fortran** und **pascal** verändern die unmittelbar rechts davon stehenden Größen.
2. Die speziellen Schlüsselwörter **near** und **far** können in Deklarationen mit den Schlüsselwörtern **fortran** und **pascal** verwendet werden. Die Folgen **fortran far** und **far fortran** sind identisch.

Die Schlüsselwörter **pascal** und **fortran** haben genau dieselbe Auswirkung auf das Programm; die Verwendung des einen oder des anderen ergibt, abgesehen von Dokumentationszwecken, keinen Unterschied. Verwenden Sie eines der beiden Schlüsselwörter, um eine BASIC-Routine zu deklarieren.

Beispiele

Die folgenden Beispiele erläutern die oben für die Schlüsselwörter **fortran** und **pascal** dargestellten Syntaxregeln.

Das folgende Beispiel deklariert `ding` als BASIC-, Pascal- oder FORTRAN-Funktion, die zwei **short**-Parameter übernimmt und einen **short**-Wert zurückgibt.

```
extern short pascal ding(short, short);   /* Beispiel 1 */
```

Das nächste Beispiel deklariert `ding` als Zeiger auf eine BASIC-, Pascal- oder FORTRAN-Routine, die einen **long**-Parameter übernimmt und keinen Wert zurückgibt. Das Schlüsselwort **void** ist passend, wenn die aufgerufene Routine eine BASIC-**SUB**-Prozedur ist, da das Schlüsselwort anzeigt, daß kein Rückgabewert erwartet wird.

```
extern void (fortran *ding) (long);   /* Beispiel 2 */
```

Das nächste Beispiel deklariert `ding` als eine **near**-BASIC-, -Pascal- oder -FORTRAN-Routine. Die Routine empfängt einen **double**-Parameter als Referenz (weil es einen Zeiger auf einen **double** erwartet) und gibt einen **short**-Wert zurück.

```
extern short near pascal ding(double *);   /* Beispiel 3 */
```

Das folgende Beispiel ist identisch zu dem vorhergehenden (`pascal near` ist identisch mit `near pascal`).

```
extern short pascal near ding(double *);   /* Beispiel 4 */
```

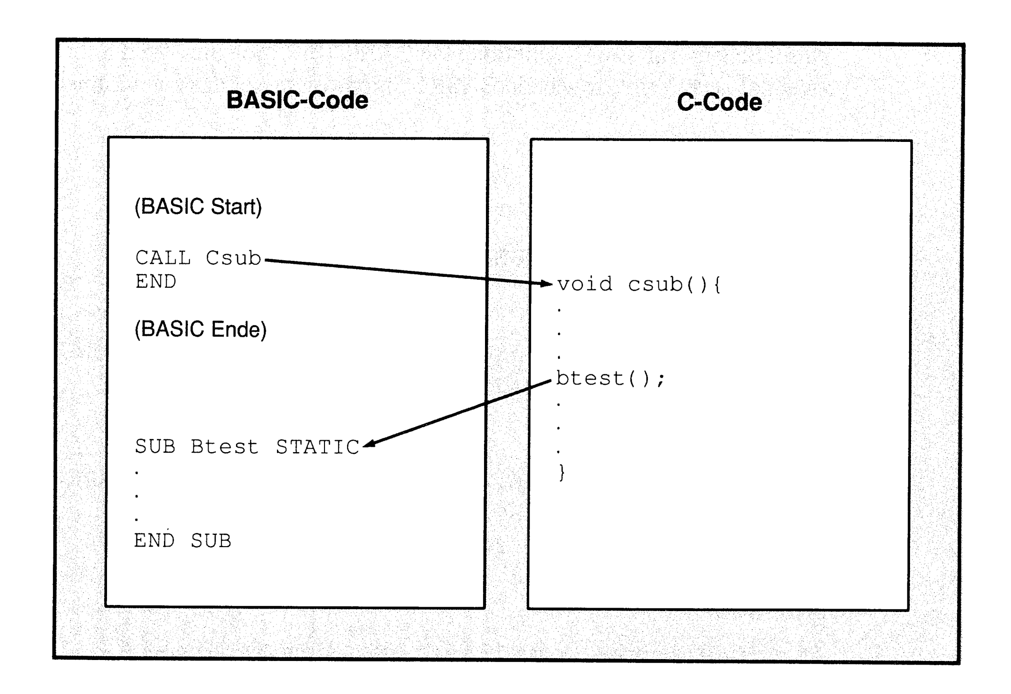
C.4.2 Aufrufe aus C zu BASIC

Wenn Sie BASIC aufrufen, müssen Sie für die Durchführung des Aufrufs die BASIC/FORTRAN/Pascal-Vereinbarungen verwenden. Solange das Hauptmodul eines Programmes nicht in BASIC vorliegt, kann keine BASIC-Routine ausgeführt werden, weil eine BASIC-Routine es erfordert, daß die Umgebung auf eine Art und Weise initialisiert ist, wie dies nur für BASIC gilt. Keine andere Sprache führt diese spezielle Initialisierung durch.

Es ist jedoch für ein Programm möglich, in BASIC zu starten, dann eine C-Funktion aufzurufen, die den Hauptteil der Arbeit des Programmes erledigt, und anschließend, falls notwendig, BASIC-**SUB**- und -**FUNCTION**-Prozeduren aufzurufen. Abbildung C.3 erläutert, wie dies durchgeführt werden kann.

Hinweis Für Programme innerhalb der QuickBASIC-Umgebung müssen alle anderssprachigen Aufrufe zu BASIC in einer Quick-Bibliothek abgelegt sein. Mit anderen Worten, eine C-Funktion kann eine BASIC-Prozedur innerhalb der Quick-Bibliothek aufrufen, sie kann aber keine Prozedur aufrufen, die innerhalb der QuickBASIC-Umgebung selbst definiert ist.

Abbildung C.3 C-Aufruf zu BASIC



Die folgenden Regeln werden empfohlen, wenn Sie BASIC aus C heraus aufrufen:

1. Beginnen Sie mit einem BASIC-Hauptmodul. Sie müssen QuickBASIC Version 4.0 oder größer einsetzen. Außerdem müssen Sie die Anweisung **DECLARE** einsetzen, um eine Schnittstelle für das C-Modul bereitzustellen. (Weitere Informationen finden Sie in Abschnitt C.3, "BASIC-Aufrufe zu C".)
2. In dem C-Modul müssen Sie die BASIC-Routine als **extern** deklarieren und Typinformationen für Parameter einfügen. Verwenden Sie entweder das Schlüsselwort **fortran** oder das Schlüsselwort **pascal** in der C-Deklaration für die BASIC-Prozedur, um die Standard-C-Aufrufvereinbarung zu überschreiben.
3. Stellen Sie sicher, daß alle Daten als kurze Zeiger übergeben werden. BASIC kann Daten auf mehrere Arten *übergeben*, ist aber nicht in der Lage, Daten in anderer Form als mit kurzer Referenz zu empfangen.

   Bei der Verwendung kurzer Zeiger geht das Programm davon aus, daß sich die Daten im Standard-Datensegment befinden. Wenn Sie Daten übergeben möchten, die sich *nicht* im Standard-Datensegment befinden, dann müssen Sie diese Daten zunächst in eine Variable kopieren, die sich im Standard-Datensegment befindet (dieser Punkt gilt nur für Programme mit dem Large-Modell).
4. Kompilieren Sie das C-Modul mit dem Medium- oder Large-Modell.

Beispiel

Das untere Beispiel demonstriert ein BASIC-Programm, das eine C-Funktion aufruft. Anschließend ruft die C-Funktion eine BASIC-Funktion auf, die die ihr übergebene Zahl zweimal zurückgibt, sowie ein BASIC-Unterprogramm, das zwei Zahlen ausgibt.

```
' BASIC-Quellcode
'
DEFINT A-Z
DECLARE SUB Cprog CDECL()
CALL Cprog
END
' Dies ist die in cprog() aufgerufene BASIC-FUNCTION.
FUNCTION Dpl (N) STATIC
   Dpl = N*2
END FUNCTION
' Dies ist die in cprog() aufgerufene BASIC-SUB.
SUB Druckzahl(A,B) STATIC
   PRINT "Die erste Zahl ist ";A
   PRINT "Die zweite Zahl ist ";B
END SUB
/* C-Quellcode, kompiliert mit dem Medium-*/
/*  oder Large-Modell */
extern int fortran dpl(int near *);
extern void fortran druckzahl(int near *, int near *);
void cprog()
{
        int near a = 5; /* NEAR stellt sicher, daß */
                        /* die Daten im Standard- */
        int near b = 6; /* Datensegment (DGROUP) */
                        /* plaziert werden */
         printf("Zweimal 5 ist %d\n", dpl(&a));
         druckzahl(&a, &b);
}
```

Beachten Sie in dem obigen Beispiel, daß die *Adressen* von `a` und `b` übergeben werden, da BASIC es erwartet, für Parameter Adressen zu empfangen. Beachten Sie auch, daß das Schlüsselwort **near** verwendet wird, um in der Deklaration der C-Funktion `druckzahl` jeden Zeiger zu deklarieren; dieses Schlüsselwort ist überflüssig, wenn feststeht, daß das C-Modul mit dem Medium- und nicht mit dem Large-Modell kompiliert ist.

In der BASIC-Deklaration von `Cprog` werden die Aufruf- und Benennungsvereinbarungen mit dem Schlüsselwort **CDECL** und in der C-Deklaration von `dpl` und `druckzahl` mit **fortran** aufgelöst.

C.5 Daten als Referenz oder Wert übergeben

Der Abschnitt C.3.1.3 führte die grundlegenden Konzepte zur Übergabe als Referenz und Übergabe als Wert ein. Es wurde auch darauf hingewiesen, daß BASIC standardmäßig als Referenz und C standardmäßig als Wert übergibt.

Dieser Abschnitt beschreibt für jede Sprache die Möglichkeiten, den Standard zu überschreiben. Zum Beispiel veranlaßt die Verwendung des Schlüsselwortes **BYVAL** in einer **DECLARE**-Anweisung BASIC dazu, einen gegebenen Parameter als Wert und nicht als Referenz zu übergeben.

Der Abschnitt C.5.1 faßt die BASIC-Methoden für Parameterübergabe zusammen und erläutert, wie Argumente als Wert, kurze Referenz und lange Referenz übergeben werden. Abschnitt C.5.2 erläutert dieselben Probleme für C. Um eine erfolgreiche Mehrsprachen-Schnittstelle zu schreiben, müssen Sie festlegen, wie jeder Parameter von der aufrufenden Routine übergeben wird, und wie jeder Parameter von der aufgerufenen Routine empfangen wird.

C.5.1 BASIC-Argumente

Standardmäßig werden in BASIC alle Argumente als kurze Referenz übergeben.

Hinweis Jede BASIC-**SUB**- oder -**FUNCTION**-Prozedur *empfängt* Parameter immer als kurze Referenz. Der noch folgende Teil des Abschnittes beschreibt nur, wie BASIC Parameter *übergibt*.

BASIC-Argumente als Wert übergeben

Ein Argument wird als Wert übergeben, wenn die aufgerufene Routine zuerst mit einer **DECLARE**-Anweisung deklariert ist und für das Argument das Schlüsselwort **BYVAL** benutzt wird. Datenfelder und benutzerdefinierte Typen können in BASIC jedoch nicht als Wert übergeben werden.

BASIC-Argumente als kurze Referenz übergeben

Der BASIC-Standard ist Übergabe als kurze Referenz. Die Verwendung von **SEG**, **BYVAL** oder **CALLS** verändert diesen Standard.

BASIC-Argumente als lange Referenz übergeben

Wenn **CALLS** verwendet wird, eine Routine aufzurufen, übergibt BASIC jedes Argument eines Aufrufs als lange Referenz. Die Verwendung von **SEG**, um einen Parameter in einer vorhergehenden **DECLARE**-Anweisung zu verändern, veranlaßt BASIC außerdem dazu, diesen Parameter als lange Referenz zu übergeben.

Beispiele

Das folgende Beispiel übergibt das erste Argument (a%) als Wert, das zweite Argument (b%) als kurze Referenz und das dritte Argument (c%) als lange Referenz.

```
DECLARE SUB Test (BYVAL a%, b%, SEG c%)
.
.
.
CALL Test (x%, y%, z%)
```

Das nächste Beispiel übergibt jedes Argument als lange Referenz.

```
CALLS Test2 (x%, y%, z%)
```

C.5.2 C-Argumente

Standardmäßig werden in C alle Datenfelder als Referenz (Small oder Large, abhängig vom Speichermodell) und alle anderen Datentypen als Wert übergeben. C verwendet lange Datenzeiger für Compact-, Large- oder Huge-Modelle und kurze Datenzeiger für Small- und Medium-Modelle.

C-Argumente als Wert übergeben

Standardmäßig übergibt C alle Nicht-Datenfelder (alle Datentypen, die nicht ausdrücklich als Datenfelder deklariert sind) als Wert.

C-Argumente als Referenz (kurz oder lang) übergeben

In C ist die Übergabe eines Zeigers auf ein Objekt identisch mit der Übergabe als Referenz des Objektes selbst. Nachdem die Kontrolle an die aufgerufene Funktion übergeben wurde, wird jeder Referenz auf den Parameter selbst ein Stern (*) vorangestellt.

Hinweis Um einen Zeiger auf ein Objekt zu übergeben, stellen Sie dem Parameter in der Aufrufanweisung ein kaufmännisches Und-Zeichen (&) voran. Um einen Zeiger auf ein Objekt zu empfangen, stellen Sie der Parameterdeklaration einen Stern (*) voran. Im zweiten Fall kann dies bedeuten, daß einem Parameter, der bereits einen Stern (*) besitzt, ein zweiter Stern (*) hinzugefügt werden muß. Um zum Beispiel einen Zeiger als Wert zu empfangen, deklarieren Sie ihn als

```
int     *ptr;
```

um aber denselben Zeiger als Referenz zu empfangen, deklarieren Sie ihn als

```
int     **ptr;
```

Standardmäßig werden Datenfelder als Referenz übergeben.

Auswirkungen des Speichermodells auf die Größe der Referenz

In C ist kurze Referenz Standard für die Übergabe von Zeigern in Small- und Medium-Modellen. Lange Referenz ist der Standard in Compact-, Large- und Huge-Modellen.

Kurze Zeiger können mit dem Schlüsselwort **near** festgelegt werden, das die Standard-Zeigergröße überschreibt. Wenn Sie jedoch beabsichtigen, die Standard-Zeigergröße eines Parameters zu überschreiben, dann müssen Sie den Parametertyp sowohl in Funktionsdeklarationen als auch in Funktionsdefinitionen ausdrücklich deklarieren.

Lange Zeiger können mit dem Schlüsselwort **far** festgelegt werden, das die Standard-Zeigergröße überschreibt.

C.6 Numerische und Zeichenketten-Daten in mehrsprachiger Programmierung

Dieser Abschnitt beschreibt die Übergabe bzw. das Empfangen unterschiedlicher Datenarten. Die Beschreibung schließt die Unterschiede im Zeichenkettenformat sowie Methoden zur Übergabe von Zeichenketten zwischen BASIC und C ein.

C.6.1 Ganze und reelle Zahlen

Ganze und reelle Zahlen sind normalerweise die einfachsten zwischen Sprachen zu übergebenden Datenarten. In jeder Sprache wird der Typ numerischer Daten jedoch anders bezeichnet; darüber hinaus sind nicht alle Datentypen in jeder Sprache verfügbar, und ein anderer Typ muß vielleicht in manchen Fällen ersetzt werden.

Die folgende Tabelle zeigt in BASIC und C äquivalente Datentypen.

BASIC	*C*
x%	**short**
INTEGER	**int**
--	**unsigned short***
--	**unsigned**
x&	**long**
LONG	**long**
--	**unsigned long***
x!	**float****
x(Standard)	**float****
SINGLE	**float****
x#	**double**
DOUBLE	**double**
--	**unsigned char*****

* In BASIC nicht verfügbar. Eine vorzeichenbehaftete Ganzzahl kann ersetzt werden, Sie müssen aber sicherstellen, daß der Bereich nicht überschritten wird.

** C konvertiert in Zuweisungen oder bei Übergabe als Wert **float** automatisch in **double**.

*** In BASIC nicht verfügbar. C konvertiert in Zuweisungen oder bei Übergabe als Wert **char** und **unsigned char** automatisch in **int.**

Warnung Wie in obiger Tabelle angemerkt, führt C manchmal automatisch Datenkonvertierungen durch, die andere Sprachen nicht durchführen. Sie können verhindern, daß C solche Konvertierungen durchführt, indem Sie eine Variable als einzige Komponente einer Struktur deklarieren und anschließend diese Struktur übergeben. Zum Beispiel können Sie eine Variable *x* vom Typ **float** übergeben, indem Sie zunächst die Struktur

```
struct {
            float    x;
}  x_struct;
```

deklarieren.

Warnung Wenn Sie eine Variable vom Typ **char** oder **float** als Wert übergeben und diese Vorsichtsmaßnahme nicht treffen, kann die C-Konvertierung dazu führen, daß das Programm versagt.

C.6.2 Zeichenketten

Zeichenketten werden in einer Vielzahl von Formaten gespeichert. Daher sind häufig Transformationen erforderlich, um Zeichenketten zwischen Sprachen zu übergeben.

Dieser Abschnitt stellt das/die in jeder Sprache verwendete/n Zeichenkettenformat/e vor und beschreibt anschließend Methoden zur Übergabe von Zeichenketten innerhalb bestimmter Sprachenkombinationen.

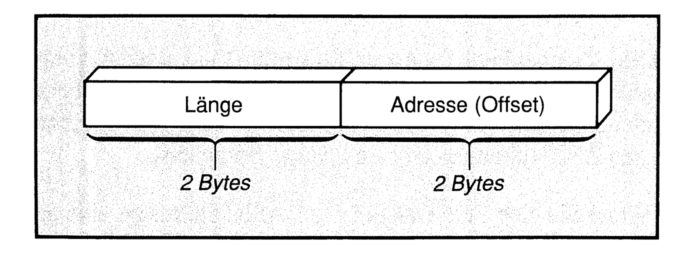
C.6.2.1 Zeichenkettenformate

Der folgende Abschnitt beschreibt, wie eine Zeichenkette in jeder Sprache gespeichert wird, und wie eine Zeichenkette als Argument übergeben wird.

BASIC-Zeichenkettenformat

In BASIC sind die meisten Zeichenketten als Vier-Byte-Zeichenkettenbeschreiber gespeichert, wie in Abbildung C.4 gezeigt.

Abbildung C.4 BASIC-Format eines Zeichenkettenbeschreibers



Das erste Feld des Zeichenkettenbeschreibers enthält eine Zwei-Byte-Ganzzahl, die die Länge des aktuellen Zeichenkettentextes angibt. Das zweite Feld enthält die Adresse dieses Textes. Diese Adresse ist ein Offset in dem Standard-Datenbereich und wird von den BASIC-Routinen zur Verwaltung des Zeichenkettenraumes zugewiesen. Diese Verwaltungsroutinen müssen verfügbar sein, um diese Adressen immer dann neu zuzuweisen, wenn sich die Länge der Zeichenkette ändert, oder der Speicher komprimiert wird; außerdem sind diese Verwaltungsroutinen nur in BASIC verfügbar. Daher sollten andere Sprachen die Länge oder die Adresse einer BASIC-Zeichenkette nicht verändern.

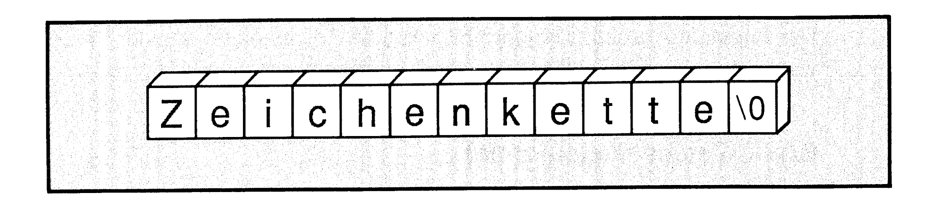
C-Zeichenkettenformat

C speichert Zeichenketten als einfache Gruppen von Bytes und verwendet ein abschließendes Nullzeichen (numerisch 0, ASCII NUL) als Begrenzer. Betrachten Sie zum Beispiel die wie folgt deklarierte Zeichenkette:

```
char str[] = "Zeichenkette"
```

Diese Zeichenkette ist in 13 Bytes des Speichers gespeichert, wie in Abbildung C.5 gezeigt.

Abbildung C.5 C-Zeichenkettenformat



Da `str` ein Datenfeld wie jedes andere ist, wird es als Referenz übergeben, genau wie andere C-Datenfelder.

C.6.2.2 BASIC-Zeichenketten übergeben

Wenn eine BASIC-Zeichenkette (wie zum Beispiel `A$`) in einer Argumentenliste erscheint, übergibt BASIC die Adresse eines Zeichenkettenbeschreibers und nicht die eigentlichen Zeichenkettendaten. Der BASIC-Zeichenkettenbeschreiber ist nicht kompatibel zu den Zeichenkettenformaten anderer Sprachen.

Warnung Wenn Sie eine Zeichenkette aus BASIC heraus an eine andere Sprache übergeben, sollte die aufgerufene Routine unter keinen Umständen die Länge oder die Adresse der Zeichenkette verändern. Anderen Sprachen fehlen BASICs Routinen zur Verwaltung des Zeichenkettenraumes. Daher dürfte die Änderung der Länge einer BASIC-Zeichenkette zur Folge haben, daß Teile des BASIC-Zeichenkettenraumes unzulässig verändert werden. Veränderungen, die sich nicht auf die Länge bzw. Adresse auswirken, sind dagegen relativ unproblematisch.

Falls die Routine, die die Zeichenkette empfängt, irgendeine BASIC-Routine aufruft, kann sich die Adresse der Zeichenkettendaten verändern. Das zweite Feld des Zeichenkettenbeschreibers wird dann mit der neuen Adresse des Zeichenkettentextes aktualisiert.

Die Funktionen **SADD** und **LEN** entnehmen Teile des Zeichenkettenbeschreibers. **SADD** entnimmt die Adresse der aktuellen Zeichenkettendaten und **LEN** entnimmt die Länge. Die Ergebnisse dieser Funktionen können anschließend an andere Sprachen übergeben werden.

BASIC sollte das Ergebnis der **SADD**-Funktion als Wert übergeben. Erinnern Sie sich daran, daß die *Adresse* der Zeichenkette, nicht die Zeichenkette selbst, als Wert übergeben wird. Dies führt dazu, daß die Zeichenkette selbst als Referenz übergeben wird. Das BASIC-Modul übergibt die Adresse der Zeichenkette, und das andere Modul empfängt die Adresse der Zeichenkette. Die von **SADD** zurückgegebene Adresse ist mit dem Typ Ganzzahl deklariert, ist aber tatsächlich gleichwertig mit einem kurzen Zeiger in C.

Übergeben Sie `LEN(A$)` genauso, wie Sie normalerweise eine Zwei-Byte-Ganzzahl übergeben. Verwenden Sie den von **SADD** zurückgegebenen Wert sofort, weil verschiedene BASIC-Operationen Bewegungen von Zeichenketten im Speicher veranlassen können. Beachten Sie, daß **SADD** nicht mit Zeichenketten fester Länge verwendet werden kann. Weitere Informationen finden Sie in Kapitel 2, "Datentypen", im *BASIC-Befehlsverzeichnis.*

C.6.2.3 BASIC-Zeichenketten an C übergeben

Bevor Sie versuchen, eine BASIC-Zeichenkette an C zu übergeben, sollten Sie vielleicht zunächst ein Nullbyte an deren Ende mit einer Anweisung wie

```
A$ = A$ + CHR$(0)
```

anfügen. Die Zeichenkette entspricht jetzt dem Zeichenkettenformat von C. Beachten Sie, daß das + Verkettungen oder Kombinationen zweier Zeichenketten anzeigt, wenn es in einem BASIC-Zeichenkettenausdruck verwendet wird.

Sie können eine der beiden folgenden Methoden verwenden, um eine Zeichenkette von BASIC an C zu übergeben:

1. Übergeben Sie die Zeichenkettenadresse und Zeichenkettenlänge als separate Argumente, indem Sie die Funktionen **SADD** und **LEN** einsetzen. Falls Sie mit einer C-Laufzeitbibliotheks-Routine binden, ist dies die einzig funktionierende Methode.

   Im folgenden Beispiel gibt `SADD(A$)` die kurze Adresse der Zeichenkettendaten zurück. Diese Adresse muß als Wert übergeben werden, da sie gleichbedeutend mit einem Zeiger ist (obwohl sie von BASIC als Ganzzahl behandelt wird). Die Übergabe als Referenz würde versuchen, die *Adresse* der Adresse, aber nicht die Adresse selbst, zu übergeben.

```
DECLARE SUB Test CDECL (BYVAL S%,BYVAL N%)
CALL Test (SADD(A$),LEN(A$))
.
.
.
void Test (s,n)
char near *s;
int n;
{
  /* Körper der Funktion */
}
```

C muß einen kurzen Zeiger empfangen, da BASIC nur die kurze Adresse (Offset) übergibt. Kurzer Zeiger ist die Standardgröße der Zeiger in den Medium-C-Modellen.

2. Übergeben Sie den Zeichenkettenbeschreiber selbst mit einer Aufrufanweisung, wie zum Beispiel

```
CALL Test2(A$)
```

In diesem Fall muß die C-Funktion für den Parameter eine Struktur deklarieren, die die passenden Felder (Adresse und Länge) für einen BASIC-Zeichenkettenbeschreiber besitzt. Die C-Funktion sollte dann erwarten, einen Zeiger auf eine Struktur dieses Typs zu empfangen. Eine solche Strukturdeklaration sowie Funktionsdefinition könnte wie folgt aussehen:

```
struct bas_zk{    /* Strukturdeklaration */
                char near *zb_addr ;
                int        zb_len  ;
               } ;
test (basic_zkett)     /* Funktionsdefinition */
     struct bas_zk *basic_zkett ;
{
/* Körper der Funktion */
}
```

Beachten Sie, daß Aufrufe innerhalb der obigen C-Funktion zurück zu BASIC die Information des Zeichenkettenbeschreibers ungültig werden lassen könnte.

C.6.2.4 C-Zeichenketten übergeben

Wenn in einer Argumentenliste eine C-Zeichenkette erscheint, übergibt C die Adresse dieser Zeichenkette. (Eine C-Zeichenkette ist lediglich ein Datenfeld und wird daher als Referenz übergeben.) C kann Daten an BASIC in Form eines Zeichenkettenbeschreibers übergeben.

C-Zeichenketten an BASIC übergeben

Um eine C-Zeichenkette an BASIC zu übergeben, müssen Sie zunächst in C eine Zeichenkette einrichten und anschließend eine Struktur anlegen, die identisch mit einem BASIC-Zeichenkettenbeschreiber ist. Übergeben Sie diese Struktur mit kurzer Referenz, wie in dem folgenden Beispiel:

```
char zkett[] = "ABC";
struct {
         char    *zb_addr;
         int     zb_len;
} str_beschr;
str_beschr.zb_addr = zkett;
str_beschr.zb_len = strlen(zkett);
bsub (&str_beschr);
```

Stellen Sie sicher, daß die Zeichenkette in C erstellt wird, nicht in BASIC. Andernfalls könnte BASIC versuchen, die Zeichenkette im Speicher zu bewegen.

C.7 Spezielle Datentypen

Dieser Abschnitt beschreibt spezielle Datentypen, die entweder Datenfelder oder strukturierte Typen (das heißt Datentypen, die mehr als ein Feld enthalten) sind.

C.7.1 Datenfelder

Wenn Sie in nur einer Sprache programmieren, stellen Datenfelder keine besonderen Probleme dar; die Sprache ist in sich geschlossen in der Behandlung von Datenfeldern. Wenn Sie jedoch mit mehr als einer Sprache programmieren, müssen Sie sich zweier spezieller Probleme bewußt sein, die bei Datenfeldern auftreten können.

1. Datenfelder sind in BASIC anders implementiert (verwirklicht) als in anderen Microsoft-Sprachen, so daß Sie besondere Vorkehrungen treffen müssen, wenn Sie ein Datenfeld von BASIC an eine andere Sprache (einschließlich Assembler) übergeben.
2. Datenfelder werden in jeder Sprache unterschiedlich deklariert und indiziert.

Dieser Abschnitt beschreibt diese Probleme der Reihe nach.

C.7.1.1 Datenfelder aus BASIC heraus übergeben

Die meisten Microsoft-Sprachen erlauben es Ihnen, direkt auf Datenfelder Bezug zu nehmen. Zum Beispiel ist in C ein Datenfeldname identisch mit der Adresse des ersten Elementes. Diese einfache Vereinbarung ist möglich, weil sich die Dateilage für Daten eines Datenfeldes nie ändert.

BASIC dagegen verwendet einen *Datenfeldbeschreiber*, der in mancher Beziehung ähnlich zu einem BASIC-Zeichenkettenbeschreiber ist. Der Datenfeldbeschreiber ist notwendig, weil BASIC die Dateilage von Datenfelddaten im Speicher verschieben kann, da BASIC Speicherzuweisung für Datenfelder dynamisch behandelt. Spezielle Informationen und Vorsichtsmaßregeln zur Übergabe von BASIC-Datenfeldern finden Sie in Kapitel 2, "Datentypen", im *BASIC-Befehlsverzeichnis*.

C besitzt kein Gegenstück zu dem BASIC-Datenfeldbeschreiber. Weit wichtiger ist, daß C der Zugriff auf die BASIC-Verwaltungsroutinen des Speicherplatzes für Datenfelder fehlt. Daher können Sie Datenfelder aus BASIC heraus *nur dann* sicher übergeben, wenn Sie die folgenden drei Regeln beachten:

1. Da nahezu alle Datenfelder (mit der Ausnahme von Zeichenketten variabler Länge) lange Daten darstellen, sollte bei deren Übergabe immer **VARSEG** verwendet werden. Um die lange Adresse eines Datenfeldes zu übergeben, sollten Sie sowohl die Funktion **VARPTR** als auch die Funktion **VARSEG** einsetzen und jedes Ergebnis als Wert übergeben. Die empfangende Sprache erhält die Adresse des ersten Elementes und betrachtet diese als Adresse des gesamten Datenfeldes. Danach kann die empfangende Sprache mit ihrer normalen Syntax für Datenfeldindizierung auf das Datenfeld zugreifen.

   Alternativ dazu kann eine lange Referenz auf ein Datenfeld übergeben werden, indem dessen erstes Element, wie in dem folgenden Beispiel, mit langer Referenz übergeben wird:

   ```
   CALLS Proz1 (A(0,0))
   CALL Proz2 (SEG A(0,0))
   ```

2. Die Routine, die das Datenfeld empfängt, darf unter keinen Umständen einen Aufruf zurück zu BASIC vornehmen. Falls sie dies tut, kann sich die Dateilage des Datenfeldes ändern, und die Adresse, die an die Routine übergeben wurde, wird bedeutungslos.
3. BASIC kann jede Komponente eines Datenfeldes als Wert übergeben. Bei dieser Methode sind die obigen Vorsichtsmaßnahmen nicht notwendig.

Hinweis Sie können die Funktionen **LBOUND** und **UBOUND** für ein BASIC-Datenfeld dazu einsetzen, die obere und untere Grenze zu bestimmen und diese Ergebnisse anschließend an die andere Routine zu übergeben. Auf diese Weise muß die Größe des Datenfeldes nicht im vorhinein festgelegt werden. Weitere Informationen zu den Funktionen **LBOUND** und **UBOUND** finden Sie im *BASIC-Befehlsverzeichnis* sowie in *Programmieren in BASIC: Ausgewählte Themen.*

C.7.1.2 Wie Sie Datenfelder indizieren und deklarieren

Jede Sprache unterscheidet sich leicht in der Art und Weise, mit der Datenfelder deklariert und indiziert werden.

Datenfelder indizieren

Die Indizierung von Datenfeldern ist lediglich eine Vereinbarung auf Quellcode-Ebene und betrifft nicht die Umwandlung von Daten. Es gibt zwei Unterschiede in der Art, mit der Elemente in unterschiedlichen Sprachen indiziert werden:

- Untere Grenzen.

  BASIC und C geben dem ersten Element eines Datenfeldes den Index 0. BASIC gibt dem Benutzer die Möglichkeit, untere Grenzen mit beliebigen Ganzzahlwerten anzugeben.
- Zeilenweise gegen spaltenweise Anordnung.

  Dieser Punkt betrifft nur Datenfelder mit mehr als einer Dimension. Bei zeilenweiser Anordnung (verwendet von C) verändert sich die Spaltendimension (dargestellt von dem äußerst rechten Index) am schnellsten. Bei spaltenweiser Anordnung (verwendet von BASIC) verändert sich die Zeilendimension (dargestellt von dem äußerst linken Index) am schnellsten. Daher sind in C die ersten vier Elemente eines Datenfeldes, dessen letztes Element `X [2][2]` ist:

  ```
  X [0][0]  X [0][1]  X [0][2]  X [1][0]
  ```

In BASIC sind die entsprechenden vier Elemente:

```
X(0,0)  X(1,0)  X(2,0)  X(0,1)
```

Genauso beziehen sich die beiden folgenden Ausdrücke auf denselben Platz eines Datenfeldes im Speicher:

```
arr1[2][8]     /* in C */
Arr1(8,2)       'in BASIC
```

Die obigen Beispiele gehen davon aus, daß sowohl das C- als auch das BASIC-Datenfeld die untere Grenze 0 verwenden.

Datenfelder deklarieren

Die folgende Tabelle zeigt Entsprechungen für Datenfelddeklarationen in jeder Sprache. In dieser Tabelle ist *z* die Anzahl der Elemente der Zeilendimension, und *s* ist die Anzahl der Elemente der Spaltendimension.

Sprache	*Datenfeld-deklaration*	*Hinweis*
BASIC	**DIM** *x*(*z-1*,*s-1*)	Mit standardmäßigen Untergrenzen von 0
C	*type x*[*z*][*s*]	Wenn als Referenz übergeben

Die Beschreibung von Datenfeldern mit Zeilen und Spalten ist eine einfache Analogie zur Betrachtungsweise von 2-dimensionalen Datenfeldern. Das Format der in obiger Tabelle gezeigten Deklarationen kann jedoch auf eine beliebige Anzahl von Dimensionen, die Sie benutzen möchten, ausgedehnt werden. Zum Beispiel entsprechen sich die C-Deklaration

```
int   arr1[2][10][15][20] ;
```

und die BASIC-Deklaration

```
DIM   Arr1%(19, 14, 9, 1)
```

Die Deklarationen entsprechen sich in dem Sinne, daß sich das von `arr1[i][j][k][l]` dargestellte Element auf dieselbe Dateilage im Speicher bezieht wie das BASIC-Datenfeldelement `Arr1(l,k,j,i)`.

Hinweis In BASIC stellen die in der Datenfelddeklaration `DIM Arr%(5,5)` verwendeten Konstanten die oberen Grenzen des Datenfeldes dar. Daher ist das letzte Element mit `Arr%(5,5)` indiziert. Die in C-Datenfelddeklarationen verwendeten Konstanten stellen die aktuelle Anzahl von Elementen in jeder Dimension dar, nicht wie in BASIC die oberen Grenzen. Daher ist das letzte Element in dem C-Datenfeld, das als `int arr[5][5]` deklariert ist, mit `arr[4][4]` und nicht mit `arr[5][5]` indiziert.

Wenn Sie ein BASIC-Programm mit dem Befehl **bc** kompilieren, können Sie die Kompilieroption **/r** wählen, die festlegt, daß die zeilenweise und nicht die spaltenweise Anordnung verwendet werden soll. Innerhalb der QuickBASIC-Umgebung ist diese Option nicht verfügbar.

C.7.2 Das Speichern von Strukturen und benutzerdefinierten Typen

Der C-Typ **struct** und der benutzerdefinierte BASIC-Typ sind gleichwertig. Diese Datentypen können daher zwischen C und BASIC übergeben werden.

Die Speichermethode kann jedoch Auswirkungen auf diese Typen haben. Standardmäßig verwendet C Wortausrichtung (ungepackter Speicher) für alle Daten außer bytegroßen Objekten sowie Datenfeldern bytegroßer Objekte. Diese Speichermethode legt fest, daß von Fall zu Fall Bytes als Füller hinzugefügt werden, so daß Wort- und Doppelwortobjekte auf einer geraden Adresse beginnen. (Zusätzlich beginnen alle verschachtelten Strukturen und Verbunde auf einer Wortgrenze.) Abbildung C.6 zeigt den Unterschied zwischen gepacktem und wortausgerichtetem Speicher:

Abbildung C.6 Das Speichern von Strukturen und Verbunden

BASIC packt Strukturen. Daher ist es wichtig, daß eine aufrufende Routine und die aufgerufene Routine in der Speichermethode übereinstimmen, wenn eine Struktur zwischen BASIC und C übergeben wird. Andernfalls werden Daten nicht richtig empfangen. Die einfachste Möglichkeit, Kompatibilität zwischen Sprachen sicherzustellen, besteht darin, für C-Module Packen einzuschalten. Die gepackte Speichermethode kann die Ausführungsgeschwindigkeit verringern, hat aber den Vorteil, daß kleinere ausführbare Dateien erzeugt werden.

C.8 Zeiger, Adreßvariablen und Common-Blöcke

Anstatt Daten direkt zu übergeben, möchten Sie vielleicht die *Adresse* eines Datums übergeben. Die Übergabe der Adresse bedeutet, daß die Daten selbst als Referenz übergeben werden. In einigen Fällen, wie zum Beispiel bei BASIC-Datenfeldern (siehe Abschnitt C.7.1.1), ist die Übergabe einer Adresse die einzige Möglichkeit, bestimmte Datenarten zwischen zwei Sprachen gemeinsam zu nutzen.

BASIC besitzt keinen formalen Adreßtyp. BASIC bietet jedoch Möglichkeiten, Adressen zu speichern und zu übergeben.

BASIC-Programme können mit **VARSEG** auf die Segmentadresse und mit **VARPTR** auf die Offsetadresse einer Variablen zugreifen. Die von diesen eingebauten Funktionen angegebenen Werte sollten dann als normale Ganzzahlvariable übergeben oder gespeichert werden. Falls Sie diese an eine andere Sprache übergeben, geschieht dies als Wert. Andernfalls unternehmen Sie den Versuch, die *Adresse* einer Adresse und nicht die Adresse selbst zu übergeben. Verwenden Sie die von **VARSEG** und **VARPTR** zurückgegebenen Werte sofort, weil BASIC-Operationen zu Bewegungen von Variablen im Speicher führen können. Weitere Informationen finden Sie in Kapitel 2, "Datentypen", im *BASIC-Befehlsverzeichnis*.

Übergeben Sie nur den Offset, um eine kurze Adresse zu übergeben; falls Sie eine lange Adresse übergeben müssen, müssen Sie Segment und Offset vielleicht einzeln übergeben. Übergeben Sie die Segmentadresse zuerst, es sei denn, es wird **CDECL** eingesetzt.

C.8.1 Common-Blöcke

Einzelne Komponenten eines BASIC-Common-Blockes können Sie in einer Argumentenliste genauso wie irgendein beliebiges Datum übergeben. Sie können einem anderssprachigen Modul aber auch sofortigen Zugriff auf den gesamten Common-Block geben.

C-Module können sich auf die Komponenten eines Common-Blockes beziehen, indem diese zunächst eine Struktur oder einen Verbund mit Feldern deklarieren, die den Variablen des Common-Blocks entsprechen.

Nachdem das C-Modul eine Struktur oder einen benutzerdefinierten Typ mit den entsprechenden Feldern definiert hat, muß es mit dem Common-Block selbst verknüpft werden.

Übergeben Sie einfach die Adresse der ersten Variablen des Blockes, um die Adresse eines Common-Blockes zu übergeben. (Mit anderen Worten, übergeben Sie die erste Variable als Referenz.) Das empfangende C-Modul sollte so gestaltet sein, daß es eine Struktur als Referenz empfängt.

Beispiel

In dem folgenden Beispiel empfängt die C-Funktion `initcb` die Adresse der Variablen `n`, die die Funktion als Zeiger auf eine Struktur mit drei Feldern behandelt:

```
'   BASIC-QUELLCODE
'
       COMMON /Cblock/n%,x#,y#
       DECLARE SUB INITCB (n%)
.
.
.
       CALL INITCB (n%)
.
.
.
/* C-Quellcode */
struct block_typ {
       int      n;
       double   x;
       double   y;
};
void initcb (block_kopf)
struct block_typ *block_kopf;
{
    block_kopf->n = 1;
    block_kopf->x = 10.0;
    block_kopf->y = 20.0;
}
```

C.8.2 Wie Sie eine variierende Anzahl von Parametern verwenden

Einige C-Funktionen, besonders hervorzuheben ist **printf**, können bei jedem Aufruf mit einer unterschiedlichen Zahl an Argumenten aufgerufen werden. Um eine solche Funktion aus einer anderen Sprache heraus aufzurufen, müssen Sie die Typüberprüfung unterdrücken, die einen Aufruf normalerweise dazu zwingt, mit einer festen Zahl an Parametern durchgeführt zu werden. In BASIC können Sie diese Typüberprüfung ausschalten, indem Sie die Parameterliste in der **DECLARE**-Anweisung weglassen, wie in Abschnitt C.3.2, "Alternative BASIC-Schnittstelle", beschrieben.

Da die Parameterzahl nicht festgelegt ist, sollte die von Ihnen aufgerufene Routine einen Mechanismus für die Festlegung, wie viele Parameter zu erwarten sind, besitzen. Häufig wird diese Information von dem ersten Parameter angezeigt. Zum Beispiel sucht die C-Funktion **printf** die als ersten Parameter übergebene Formatzeichenkette ab. Die Anzahl an Feldern in der Formatzeichenkette legt fest, wie viele zusätzliche Parameter die Funktion akzeptieren soll. Das folgende Beispiel zeigt zwei Möglichkeiten für den Aufruf der C-Bibliotheksfunktion **printf**.

Beispiele

In dem ersten Beispiel ist eine festgelegte Zahl von Argumenten deklariert, und das Schlüsselwort **BYVAL** steht vor jedem Parameter in der **DECLARE**-Anweisung, um sicherzustellen, daß die wirkliche Adresse des Zeichenkettenargumentes übergeben wird.

```
DEFINT A-Z
DECLARE SUB printf CDECL (BYVAL format, BYVAL wert)
ZKette1$ = "Wert, übergeben an die formatierte
Zeichenkette: %d"+CHR$(0)
Zahl = 19
CALL printf(SADD(ZKette1$), Zahl)
END
```

Die folgende Abwandlung des obigen Beispiels verwendet eine spezielle Form der **DECLARE**-Anweisung, um die Funktion `printf` zu deklarieren. Die Klammern, die normalerweise die formalen Parameter enthalten, sind aus der Deklaration weggelassen. Dies informiert QuickBASIC über zwei Dinge:

- Die Prozedur ist nicht in BASIC geschrieben.
- Die Prozedur kann eine variierende Zahl von Argumenten haben. Das Schlüsselwort **BYVAL** wird in dem Programm zwar noch verwendet, wird diesesmal aber mit jedem Argument in dem Aufruf zu `printf` übergeben.

In beiden Beispielen wird in der Zeichenkette, die `printf` übergeben wird, das C-Formatierungszeichen `%d` eingesetzt, das `printf` mit dem Wert des zweiten Argumentes ersetzt.

```
DEFINT A-Z
DECLARE SUB printf CDECL
ZKettel$ = "Wert, übergeben an die formatierte
Zeichenkette: %d"+CHR$(0)
Zahl = 19
CALL printf(BYVAL SADD(ZKettel$), BYVAL Zahl)
END
```

Das obige Beispiel beschreibt den BASIC-Aufruf von **printf**. Wenn Sie die von einem solchen Programm erstellte Objektdatei von der Befehlszeile aus binden, löst der Linker den Bezug auf die Funktion **printf** auf, wenn die ausführbare Datei angelegt wird. Wenn Sie **printf** jedoch innerhalb der QuickBASIC-Umgebung aufrufen möchten, müssen Sie die Funktion innerhalb einer Quick-Bibliothek zur Verfügung stellen. Die einfachste Möglichkeit, dies durchzuführen, besteht darin, eine "Dummy"-C-Funktion zu schreiben und zu kompilieren, die **printf** aufruft (stellen Sie sicher, daß Sie, wie bereits zuvor betont, mit einem Medium-Modell kompilieren). Anstatt sie mit einem BASIC-Programm auf der **link**-Befehlszeile zu binden, binden Sie die Objektdatei anschließend, wie in dem folgenden Beispiel gezeigt, in eine Quick-Bibliothek ein:

```
link /q dummy.obj, ,,bqlb40.lib;
```

In diesem Fall wird eine `dummy.qlb` benannte Quick-Bibliothek angelegt; sie enthält nur die Funktion **printf**. Falls Sie diese Bibliothek beim Aufruf von QuickBASIC laden, können Sie anschließend Aufrufe zu **printf** wie folgt durchführen:

```
qb /l dummy
```

Mit dieser Methode können Sie Quick-Bibliotheken erstellen, die sowohl hilfreiche Funktionen aus der Standard-C-Bibliothek als auch von Ihnen selbst geschriebene anderssprachige Routinen enthalten. Weitere Informationen finden Sie in Kapitel 8, "Quick-Bibliotheken".

C.9 Die Routine B_OnExit

Sie können **B_OnExit** verwenden, wenn anderssprachige Routinen spezielle Aktionen durchführen, die vor der Programmbeendigung oder dem Neustart des Programmes rückgängig gemacht werden müssen. Zum Beispiel läuft ein innerhalb der QuickBASIC-Umgebung ausgeführtes Programm, das anderssprachige Routinen in einer Quick-Bibliothek aufruft, vielleicht nicht immer bis zur normalen Beendigung. Falls solche Routinen vor der Beendigung besondere Aktionen (zum Beispiel Zurücksetzen eines zuvor installierten Interruptvektors) durchführen müssen, können Sie mit dem Einbinden eines Aufrufs zu **B_OnExit** in die Routine sicherstellen, daß Ihre Beendigungsroutine immer aufgerufen wird. Das folgende Beispiel demonstriert einen solchen Aufruf (der Einfachheit halber fehlt der Funktion Fehlerbehandlungs-Code). Beachten Sie, daß eine solche Funktion in C mit dem Large-Modell kompiliert wird.

```
#include <malloc.h>
extern pascal far B_OnExit(); /* Routine dekla- */
                              /* rieren */
int *p_GanzFeld;
void InitProz()
{
   void EndeProz();       /* Funktion EndeProz */
                          /* deklarieren */
   /* Weise langen Speicherplatz für Datenfeld */
   /* mit 20 Ganzzahlen zu: */
   p_GanzFeld = (int *)malloc(20*sizeof(int));
   /* Melde Beendigungsroutine (EndeProz) für */
   /* BASIC an: */
   B_OnExit(EndeProz);
}
                  /* Die Funktion EndeProz wird vor */
void EndeProz ()  /* jedem Neustart bzw. vor jeder */
                  /* Beendigung des Programmes */
                  /* aufgerufen. */
{
   free(p_GanzFeld);  /* Gibt langen Speicherplatz */
                      /* wieder frei, der zuvor von */
                      /* InitProz zugewiesen wurde. */
}
```

Falls sich die Funktion `InitProz` in einer Quick-Bibliothek befände, würde der Aufruf von **B_OnExit** sicherstellen, daß der in dem Aufruf von **malloc** reservierte Speicherplatz freigegeben wird, wenn das Programm enden sollte, bevor die normale Beendigung ausgeführt werden konnte. Die Routine könnte mehrmals aufgerufen werden, da das Programm aus der QuickBASIC-Umgebung heraus mehrmals ausgeführt werden kann. Die Funktion `EndeProz` selbst würde jedoch nur einmal bei jeder Programmausführung aufgerufen.

Das folgende BASIC-Programm ist ein Beispiel für den Aufruf der Funktion `InitProz`:

```
DECLARE SUB InitProz CDECL
X = SETMEM(-2048)       ' Schaffe Platz für die malloc-
                        ' Speicherzuweisung in der
                        ' C-Funktion.
CALL InitProz
END
```

Falls mehr als 32 Routinen registriert sind, gibt **B_OnExit** NULL zurück und zeigt damit an, daß nicht genügend Speicherplatz vorhanden ist, um die aktuelle Routine einzutragen. (Beachten Sie, daß **B_OnExit** dieselben Rückgabewerte hat wie die Microsoft-C-Laufzeit-Bibliotheksroutine **onexit.**)

B_OnExit kann mit Assembler oder anderen beliebigen anderssprachigen Routinen, die Sie in einer Quick-Bibliothek ablegen, benutzt werden. Für Programme, die komplett von der Befehlszeile aus kompiliert und gebunden sind, ist **B_OnExit** optional.

C.10 Assembler/BASIC-Schnittstelle

Mit dem Microsoft Macro Assembler (MASM) können Sie Assembler-Routinen schreiben, die mit Modulen gebunden werden können, die mit Microsoft-BASIC entwickelt wurden. Dieser Abschnitt beschreibt die empfohlenen Richtlinien für die Programmierung, um zu Microsoft-QuickBASIC kompatible Assembler-Routinen zu schreiben.

Das Schreiben von Assembler-Routinen für Microsoft-Hochsprachen ist am einfachsten, wenn Sie die vereinfachten Segment-Direktiven verwenden, die von dem Microsoft Macro Assembler Version 5.0 unterstützt werden. Allgemein wird in diesem Handbuch davon ausgegangen, daß Sie Version 5.0 besitzen. Um Informationen darüber zu erhalten, wie Sie Assembler-Routinen ohne die vereinfachten Segment-Direktiven schreiben, schlagen Sie in Abschnitt C.10.4 nach, in dem Sie die Anweisungen **SEGMENT**, **GROUP** und **ASSUME** finden.

C.10.1 Die Assembler-Prozedur schreiben

Sie können Assembler-Prozeduren aufrufen, indem Sie im wesentlichen dieselben Vereinbarungen wie für Compiler erzeugten Code verwenden. Dieser Abschnitt beschreibt, wie Sie diese Vereinbarungen einsetzen, um Assembler-Prozeduren aufzurufen. Prozeduren, die diese Vereinbarungen beachten, können rekursiv aufgerufen und effektiv mit der Stack Trace-Eigenschaft des Microsoft CodeView-Debuggers eingesetzt werden, der mit Microsoft Macro Assembler Version 5.0 und Microsoft C Version 5.0 geliefert wird.

Die Standardmethode für Assembler-Schnittstellen besteht aus diesen Schritten:

- Einrichten der Prozedur
- Beginn der Prozedur
- Lokale Daten zuweisen (optional)
- Registerwerte sichern
- Auf Parameter zugreifen
- Einen Wert zurückgeben (optional)
- Die Prozedur verlassen

Die Abschnitte C.10.1.1 bis C.10.1.8 beschreiben jeden dieser Schritte.

Hinweis Die Originaldisketten des Microsoft Macro Assemblers Version 5.0 enthalten die Include-Datei **MIXED.INC**, die automatisch viele der schwierigen Aufgaben der Stapelverwaltung übernimmt, die in den nächsten Abschnitten beschrieben werden. Die Dokumentation für **MIXED.INC** finden Sie in der Datei **MIXED.DOC**, die sich ebenfalls auf den Originaldisketten der Version 5.0 befindet. Spätere Versionen können diese Informationen in den Handbüchern enthalten.

C.10.1.1 Die Prozedur einrichten

Der Linker kann die Assembler-Prozedur solange nicht mit dem aufrufenden Programm kombinieren, wie keine kompatiblen Segmente benutzt werden bzw. die Prozedur selbst nicht richtig deklariert ist. Die folgenden vier Punkte sind wahrscheinlich hilfreich.

1. Wenn Sie die Version 5.0 des Microsoft Macro Assemblers besitzen, schreiben Sie die **.MODEL**-Direktive an den Anfang der Quelldatei. Diese Direktive veranlaßt, daß automatisch die richtigen Rücksprungarten erzeugt werden (**NEAR** für das Small- oder Compact-Modell, andernfalls **FAR**). Aus BASIC heraus aufgerufene Module sollten **.MODEL MEDIUM** sein. Falls Sie eine Version des Macro Assemblers kleiner als 5.0 einsetzen, deklarieren Sie die Prozedur **FAR**.
2. Wenn Sie Version 5.0 oder größer des Microsoft Macro Assemblers besitzen, verwenden Sie die vereinfachte Segment-Direktive **.CODE**, um das Code-Segment zu deklarieren, und **.DATA**, um das Daten-Segment zu deklarieren. (Wenn Sie keine Datendeklarationen haben, genügt das Vorhandensein eines Code-Segmentes.) Falls Sie eine frühere Version des Assemblers einsetzen, schauen Sie sich die Direktiven **SEGMENT**, **GROUP** und **ASSUME** in Abschnitt C.10.4, "Das Microsoft-Segmentmodell", an.
3. Die Prozedurmarke muß mit der Direktive **PUBLIC** global deklariert werden. Diese Deklaration ermöglicht es der Prozedur, von anderen Modulen aufgerufen zu werden. Außerdem müssen alle Daten, die Sie global zu anderen Modulen machen möchten, als **PUBLIC** deklariert werden.
4. Globale Daten oder Prozeduren, auf die die Routine zugreift, müssen **EXTRN** deklariert werden. Die sicherste Art, **EXTRN**-Code-Deklarationen zu verwenden, besteht darin, die Direktive außerhalb irgendeiner Segment-Definition zu plazieren.

C.10.1.2 Beginn der Prozedur

Die Prozedur beginnt mit zwei Anweisungen:

```
push    bp
mov     bp,sp
```

Diese Anweisungen legen **BP** als "Rahmenzeiger" (Frame Pointer) fest. Der Rahmenzeiger wird verwendet, um auf Parameter und lokale Daten zuzugreifen, die auf dem Stapel abgelegt sind. Der Wert des Basisregisters **BP** bleibt überall in der Prozedur konstant, so daß jeder Parameter als fester Versatz zu **BP** adressiert werden kann. (**SP** kann nicht für diesen Zweck verwendet werden, weil es kein Index- oder Basisregister darstellt. Außerdem kann sich der Wert von **SP** ändern, sobald mehr Daten auf dem Stapel abgelegt werden.)

Die Anweisung `push bp` rettet den Wert von **BP**. Dieser Wert wird von der aufrufenden Prozedur benötigt, sobald die aktuelle Prozedur beendet wird. Die Anweisung `mov bp,sp` liest den Wert, den der Stapelzeiger zum Zeitpunkt des Eingangs in die Prozedur hatte (und legt damit die Adressierbarkeit von Parametern fest).

C.10.1.3 Lokale Daten zuweisen (optional)

Eine Assembler-Prozedur kann zur Implementierung lokaler Daten dieselbe Technik verwenden, die auch von höheren Sprachen (High Level Language) eingesetzt wird. Um Speicherplatz für lokale Daten einzurichten, müssen Sie nur den Inhalt von **SP** in der dritten Prozeduranweisung verkleinern. (Um eine korrekte Ausführung sicherzustellen, sollten Sie **SP** immer um einen geraden Betrag vergrößern oder verkleinern.) Die Verkleinerung von **SP** reserviert für die lokalen Daten Platz auf dem Stapel. Dieser Platz muß am Ende der Prozedur wiederhergestellt werden.

```
push    bp
mov     bp,sp
sub     sp,platz
```

In dem obigen Text stellt *platz* die gesamte Größe der lokalen Daten in Bytes dar. Auf lokale Variablen wird dann mit festem negativem Versatz zu **BP** zugegriffen.

Beispiel

Das folgende Beispiel verwendet zwei lokale Variablen, die beide zwei Bytes lang sind. **SP** wird um 4 verkleinert, da insgesamt vier Bytes lokale Daten vorhanden sind. Später wird jede dieser Variablen zu Null initialisiert. Diese Variablen sind niemals mit irgendeiner Assembler-Direktive formal deklariert; der Programmierer muß diese Variablen eigenhändig verwalten. Lokale Variablen werden auch als dynamische, Stapel- oder automatische Variablen bezeichnet.

```
push    bp
mov     bp,sp
sub     sp,4
.
.
.
mov     WORD PTR [bp-2],0
mov     WORD PTR [bp-4],0
```

C.10.1.4 Registerwerte sichern

Eine Prozedur, die von irgendeiner Microsoft-Hochsprache aufgerufen wird, sollte die Werte von **SI**, **DI**, **SS** und **DS** sichern (zusätzlich zu **BP**, das bereits gesichert ist). Legen Sie daher auf dem Stapel jeden Registerwert ab, den die Prozedur verändert. Falls die Prozedur keinen der Werte dieser Register verändert, müssen die Register nicht auf dem Stapel abgelegt werden.

Die empfohlene Methode (verwendet von Hochsprachen) besteht darin, die Register zu sichern, nachdem der Rahmenzeiger gesetzt ist und der Speicherplatz für lokale Daten (falls vorhanden) zugewiesen ist.

Im folgenden Beispiel müssen **DI** und **SI** (in dieser Reihenfolge) vor dem Prozedurende vom Stapel geholt werden.

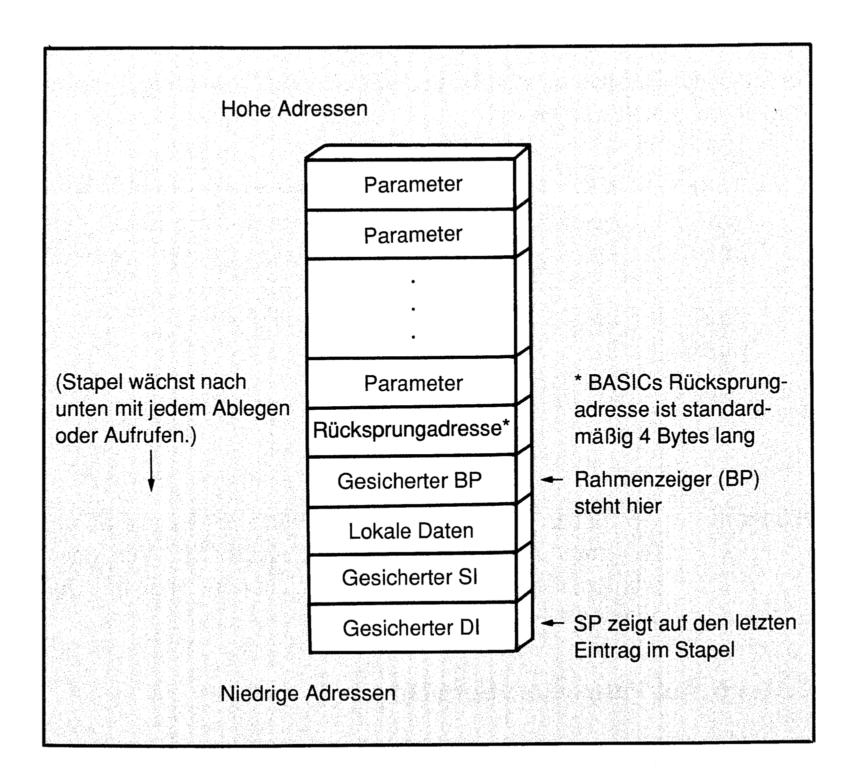
```
    push    bp      ; Alten Rahmenzeiger sichern.
    mov     bp,sp   ; Aktuellen Rahmenzeiger festlegen.
    sub     sp,4    ; Speicherplatz für lokale Daten
                    ; zuweisen.
    push    si      ; SI und DI sichern.
    push    di
.
.
.
```

Hinweis Obwohl es empfohlen wird, die Werte von **DI** und **SI** zu sichern, war dies für Prozeduren, die mit früheren QuickBASIC-Versionen verwendet wurden, nicht notwendig und ist nicht notwendig mit QuickBASIC Version 4.0.

C.10.1.5 Auf Parameter zugreifen

Nachdem Sie den Rahmenzeiger der Prozedur einmal festgelegt haben, lokalen Daten Speicherplatz zugewiesen haben (falls gewünscht) und jedes Register, das gesichert werden muß, auf dem Stapel abgelegt haben, können Sie den Hauptteil der Prozedur schreiben. Um Anweisungen zu schreiben, die auf Parameter zugreifen können, betrachten Sie das allgemeine Bild des Stapelrahmens nach einem Prozeduraufruf, wie in Abbildung C.7 dargestellt.

Abbildung C.7 Der Stapelrahmen



Der Stapelrahmen für die Prozedur wird von der folgenden Ereignisreihe festgelegt:

1. Das aufrufende Programm legt jeden Parameter auf dem Stapel ab, wonach **SP** auf den letzten abgelegten Parameter zeigt.
2. Das aufrufende Programm führt eine **CALL**-Anweisung aus, die veranlaßt, daß die Rücksprungadresse (die Stelle in dem aufrufenden Programm, zu der die Kontrolle schließlich zurückkehrt) auf dem Stapel abgelegt wird. Da BASIC immer einen **FAR**-Aufruf benutzt, ist diese Adresse vier Bytes lang. **SP** zeigt nun auf diese Adresse.
3. Die erste Anweisung der aufgerufenen Prozedur sichert den alten Wert von **BP** mit der Anweisung `push bp`. **SP** zeigt nun auf die gesicherte Kopie von **BP**.
4. **BP** wird dazu verwendet, mit der Anweisung `move bp,sp` den aktuellen Wert von **SP** zu lesen. **BP** zeigt daher auf den alten Wert von **BP**.
5. Während **BP** überall in der Prozedur konstant bleibt, kann **SP** verkleinert werden, um auf dem Stapel Platz für lokale Daten oder gesicherte Register bereitzustellen.

Allgemein ist der Versatz eines Parameters X (zu **BP**) gleich

```
2 + Länge der Rücksprungadresse
  + Gesamtgröße der Parameter zwischen X und BP
```

Stellen Sie sich zum Beispiel eine Prozedur vor, die einen Parameter, eine Zwei-Byte-Adresse, empfangen hat. Da die Länge der Rücksprungadresse in BASIC immer vier Bytes beträgt, ist der Versatz des Parameters

```
Versatz des Argumentes = 2 + Länge der Rücksprungadresse
                       = 2 + 4
                       = 6
```

Das Argument kann daher mit der folgenden Anweisung in **BX** geladen werden:

```
        mov   bx, [bp+6]
```

Sobald Sie den Versatz jedes Parameters bestimmt haben, möchten Sie vielleicht Zeichenketten-Äquivalente oder -Strukturen verwenden, so daß Sie in Ihrem Assembler-Quellcode mit einem einzigen Bezeichner auf den Parameter Bezug nehmen können. Auf den obigen Parameter bei **BP+6** kann einfach zugegriffen werden, wenn Sie die folgende Anweisung an den Beginn Ihrer Assembler-Quelldatei schreiben:

```
Arg1      EQU   [bp+6]
```

Sie können sich dann in einer beliebigen Anweisung mit `Arg1` auf diesen Parameter beziehen. Die Verwendung dieser Möglichkeit ist optional.

Hinweis Microsoft-Hochsprachen legen Segmentadressen immer auf dem Stapel ab, bevor sie die Offsetadresse auf dem Stapel ablegen. Darüber hinaus wird das höherwertige Wort immer vor dem niederwertigen Wort auf dem Stapel abgelegt, wenn Argumente, die größer als zwei Bytes sind, auf dem Stapel abgelegt werden.

Dieser Standard, nach dem Segmentadressen vor Offsetadressen auf dem Stapel abgelegt werden, erleichtert die Verwendung der Anweisungen **LES** und **LDS**.

C.10.1.6 Einen Wert zurückgeben (optional)

BASIC besitzt eine einfache Vereinbarung für den Empfang von Rückgabewerten, wenn der zurückzugebende Datentyp einfach (das heißt, kein Gleitkomma-Wert, Datenfeld oder strukturierter Typ) und nicht länger als vier Bytes ist. Dies schließt alle Zeiger und alle Parameter ein, die als Referenz übergeben werden, wie in der folgenden Liste gezeigt:

Datentyp	*Zurückgegeben in Register*
2-Byte-Ganzzahl(%)	**AX**
4-Byte-Ganzzahl(&)	Höherwertiger Teil in **DX**; niederwertiger Teil in **AX**
Alle anderen Typen	Kurzer Offset in **AX**

Eine Assembler-Prozedur, die aus BASIC heraus aufgerufen wird, muß für die Rückgabe eines Gleitkomma-Wertes, benutzerdefinierten Typs und Datenfeldes sowie ganzzahligen Wertes, der länger als 4 Bytes ist, spezielle Vereinbarungen verwenden, wie in Abschnitt C.10.1.7 erläutert.

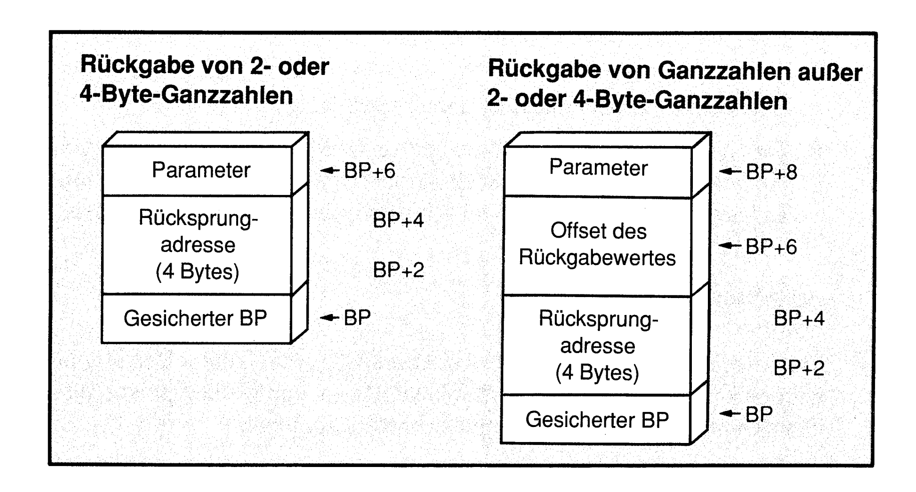
C.10.1.7 Numerische Rückgabewerte, die nicht Zwei- oder Vier-Byte-Ganzzahlen sind

Um eine Schnittstelle für numerische Rückgabewerte zu erstellen, die weder Zwei-Byte-Ganzzahlen (%) noch Vier-Byte-Ganzzahlen (&) sind, führen BASIC-Module die folgenden Schritte aus, bevor sie Ihre Prozedur aufrufen (vorausgesetzt, daß die BASIC-Deklarationen nicht das Schlüsselwort **CDECL** angeben):

1. Beim Aufruf Ihrer Prozedur wird ein zusätzlicher Parameter übergeben; dieser Parameter enthält die Offsetadresse des aktuellen Rückgabewertes. Dieser Parameter ist direkt über der Rückgabeadresse plaziert. (Mit anderen Worten ist dieser Parameter der letzte auf dem Stapel abgelegte Wert.)
2. Sowohl **SS** als auch **DS** enthalten die Segmentadresse des Rückgabewertes.

Der zusätzliche Parameter (der die Offsetadresse des Rückgabewertes enthält) wird immer bei **BP+6** abgelegt. Darüber hinaus erhöht sein Vorhandensein automatisch den Versatz aller anderen Parameter um zwei, wie in Abbildung C.8 gezeigt.

Abbildung C.8 BASIC-Rückgabewerte



Ihre Assembler-Prozedur kann andere numerische Werte als Zwei- und Vier-Byte-Ganzzahlen erfolgreich zurückgeben, wenn Sie diesen Schritten folgen:

1. Legen Sie die Daten des Rückgabewertes auf der Position ab, auf die der Offset des Rückgabewertes zeigt.
2. Kopieren Sie den Offset des Rückgabewertes (abgelegt bei **BP+6**) in **AX**. Dies ist notwendig, weil das aufrufende Modul erwartet, daß **AX** auf den Rückgabewert zeigt.
3. Verlassen Sie die Prozedur, wie im nächsten Abschnitt beschrieben.

C.10.1.8 Die Prozedur verlassen

Mehrere Schritte können bei der Beendigung der Prozedur notwendig sein:

1. Falls irgendwelche der Register **SS, DS, SI** oder **DI** gesichert wurden, müssen diese in umgekehrter Reihenfolge, in der sie gesichert wurden, vom Stapel genommen werden.
2. Falls zu Beginn der Prozedur lokalen Daten Speicherplatz zugewiesen wurde, muß **SP** mit der Anweisung

   ```
   mov sp,bp
   ```

   wiederhergestellt werden.

3. Stellen Sie **BP** mit der Anweisung

   ```
   pop bp
   ```

   wieder her. Dieser Schritt ist immer notwendig.

4. Da die BASIC-Aufrufvereinbarung verwendet wird, müssen Sie die Form `ret` *n* der Anweisung verwenden, um den Stapel mit Rücksicht auf die Parameter, die von der aufrufenden Prozedur auf dem Stapel abgelegt wurden, anzupassen.

Beispiele

Das folgende Beispiel zeigt die einfachste Möglichkeit einer Beendigungsfolge. Es wurden keine Register gesichert, lokalen Daten wurde kein Speicherplatz zugewiesen, und es wurden der Routine keine Parameter übergeben.

```
pop     bp
ret
```

Das nächste Beispiel zeigt die Beendigungsfolge einer Prozedur, die zuvor **SI** und **DI** gesichert hat, lokalen Daten Speicherplatz zugewiesen hat, die BASIC-Aufrufvereinbarung verwendete und 6 Bytes an Parametern empfangen hat. Die Prozedur muß daher `ret 6` verwenden, um die 6 Bytes für den Parameter auf dem Stapel wiederherzustellen.

```
pop     si
pop     sp,bp   ; Entferne Platz lokaler Daten.
                ; Stelle alten Rahmenzeiger
pop     bp     ; wieder her.
ret     6      ; Beende und entferne 6 Bytes
                ; für Argumente.
```

C.10.2 Aufrufe aus BASIC heraus

Ein BASIC-Programm kann eine Assembler-Prozedur einer anderen Quelldatei mit der Anweisung **CALL** bzw. **CALLS** aufrufen. Außerdem ist die richtige Verwendung der Anweisung **DECLARE** wichtig. Zusätzlich zu den in Abschnitt C.10.1, "Die Assembler-Prozedur schreiben", erläuterten Schritten, können die folgenden Richtlinien hilfreich sein:

1. Deklarieren Sie Prozeduren, die aus BASIC heraus aufgerufen werden, als **FAR**.

2. Beachten Sie die BASIC-Aufrufvereinbarungen:

   a. Parameter werden auf dem Stapel in derselben Reihenfolge plaziert, in der sie im BASIC-Quellcode erscheinen. Der erste Parameter liegt ganz oben im Speicher (weil er auch der erste auf dem Stapel abzulegende Parameter ist und der Stapel nach unten wächst).

   b. Standardmäßig werden BASIC-Parameter als Zwei-Byte-Adressen als Referenz übergeben.

   c. Beim Verlassen muß die Prozedur **SP** auf den Wert zurücksetzen, den dieser vor der Plazierung der Parameter auf dem Stapel hatte. Dies wird von dem Befehl `ret` *platz* ausgeführt, wobei *platz* die Gesamtgröße aller Parameter in Bytes ist.

3. Beachten Sie die BASIC-Benennungsvereinbarungen:

   BASIC gibt symbolische Namen in Großbuchstaben aus, genauso wie dies der Assembler standardmäßig tut. BASIC beachtet bis zu 40 Zeichen eines Namens, während der Assembler nur die ersten 31 Zeichen beachtet (dies sollte kaum Probleme verursachen).

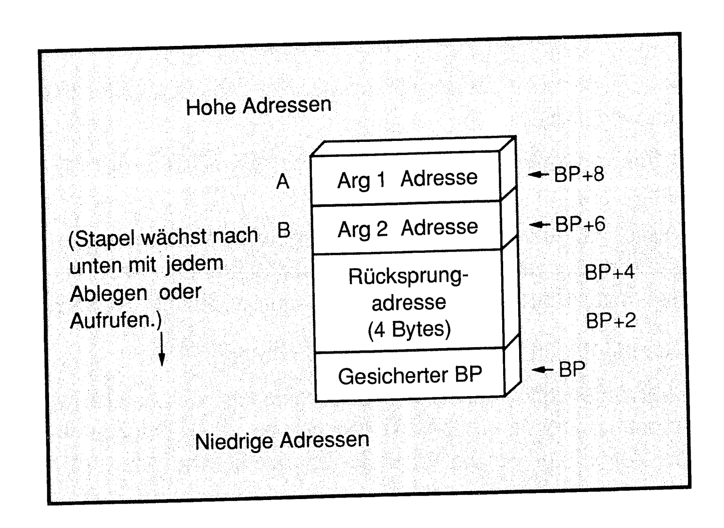
Hinweis Die QuickBASIC Originaldiskette 3 enthält eine QB.QLB benannte Quick-Bibliothek sowie eine QB.BI benannte Include-Datei. Diese Dateien enthalten mehrere Routinen, die es ermöglichen, Assembler-Routinen aus der QuickBASIC-Umgebung heraus aufzurufen. Weitere Informationen zur Verwendung dieser Bibliothek finden Sie in Kapitel 8, "Quick-Bibliotheken".

In dem folgenden Beispielprogramm ruft BASIC eine Assembler-Prozedur auf, die $A \times 2^B$ berechnet, wobei A und B der erste und zweite Parameter sind. Die Berechnung wird durchgeführt, indem die Bits in A B-mal nach links geschoben werden.

```
' BASIC-Programm.
'
DEFINT A-Z
'
DECLARE FUNCTION Hoch2(A%,B%)
'
PRINT "3 mal 2 hoch 5 ist ";
PRINT Hoch2(3,5)
END
```

Um zu verstehen, wie die Assembler-Prozedur geschrieben werden muß, erinnern Sie sich daran, wie Parameter auf dem Stapel abgelegt werden (gezeigt in Abbildung C.9):

Abbildung C.9 BASIC-Stapelrahmen



Die Rücksprungadresse ist 4 Bytes lang, weil Prozeduren, die aus BASIC heraus aufgerufen werden, **FAR** sein müssen. Arg1 (Parameter 1) liegt höher im Speicher als Arg2, da BASIC Argumente (Parameter) in derselben Reihenfolge, in der sie erscheinen, auf dem Stapel ablegt. Außerdem wird jedes Argument als Zwei-Byte-Offsetadresse, dem Standard in BASIC, übergeben.

Die Assembler-Prozedur kann wie folgt geschrieben werden:

```
.MODEL MEDIUM
.CODE
        PUBLIC  Hoch2
Hoch2   PROC
        push    bp          ; Eingangssequenz - Alten BP
                            ; sichern.
        mov     bp,sp       ; Setze Stapel-Rahmenzeiger
        mov     bx, [bp+8]  ; Lade Arg1 in
        mov     ax, [bx]    ;   AX
        mov     bx, [bp+6]  ; Lade Arg2 in
        mov     cx, [bx]    ;   CX
        shl     ax,cl       ; AX = AX * (2 hoch CX)
                            ; Rückgabewert in AX
        pop     bp          ; Alten BP wiederherstellen
        ret     4           ; Rücksprung und vier Bytes
                            ; wiederherstellen
Hoch2   ENDP
        END
```

Beachten Sie, daß jeder Parameter in zwei Schritten geladen werden muß, da die *Adresse* eines jeden Parameters und nicht sein Wert übergeben wird. Beachten Sie außerdem, daß der Stapel mit der Anweisung `ret 4` wiederhergestellt wird, da die Gesamtgröße der Parameter 4 Bytes beträgt. (Das vorhergehende Beispiel ist vereinfacht, um die zwischensprachliche Schnittstelle zu demonstrieren. Code zur Behandlung möglicher Fehler, wie zum Beispiel Überlauf, ist nicht eingefügt, ist aber ein wichtiger Gesichtspunkt für solche Prozeduren.)

C.10.3 Wie Sie CDECL in Aufrufen aus BASIC heraus einsetzen

Um eine Assembler-Routine aufzurufen, können Sie in der **DECLARE**-Anweisung Ihres BASIC-Moduls das Schlüsselwort **CDECL** einsetzen. Falls Sie dies tun, bestimmen die C-Aufrufvereinbarungen, und nicht diejenigen von BASIC, die Reihenfolge, in der die Argumente von der Assembler-Routine empfangen werden, und darüber hinaus die Art, in der Rücksprünge behandelt werden. Der Hauptvorteil bei der Verwendung von **CDECL** ist, daß Sie die Assembler-Routine dann mit einer unterschiedlichen Anzahl an Argumenten aufrufen können. Diese Technik ist analog zu dem Aufruf einer C-Funktion mit **CDECL** aus BASIC heraus (siehe Abschnitt C.3). Beachten Sie bei der Verwendung von **CDECL** die C-Aufrufvereinbarungen:

1. Parameter werden auf dem Stapel in umgekehrter Reihenfolge, in der Sie im BASIC-Quellcode erscheinen, plaziert. Das bedeutet, der erste Parameter liegt im Speicher ganz unten (da der Stapel nach unten wächst, und dieser Parameter der letzte auf dem Stapel abzulegende Parameter ist).
2. Führen Sie den Rücksprung mit einer einfachen `ret`-Anweisung aus. Stellen Sie den Stapel *nicht* mit `ret` *platz* wieder her, da die Verwendung von **CDECL** die aufrufende Routine dazu veranlaßt, den Stapel selbst wiederherzustellen, sobald sie die Kontrolle zurückerhält.

   Wenn der Rückgabewert kein numerischer Zwei-Byte- oder Vier-Byte-Wert ist, muß eine Prozedur, die von BASIC mit **CDECL** aufgerufen wird, Speicherplatz für den Rückgabewert zuweisen, und anschließend dessen Adresse in **AX** ablegen. Eine einfache Möglichkeit, Speicherplatz für den Rückgabewert zu schaffen, besteht darin, ihn in einem Datensegment zu deklarieren.
3. Falls in der BASIC-**DECLARE**-Anweisung **CDECL** verwendet wird, müssen Sie die Assembler-Prozedur mit einem führenden Unterstreichungszeichen benennen, es sei denn, Sie verwenden die Möglichkeit **ALIAS** (siehe Abschnitt C.3.1.2).

C.10.4 Das Microsoft-Segmentmodell

Wenn Sie die vereinfachten Segment-Direktiven selbst verwenden, müssen Sie die Namen, die jedem Segment zugewiesen werden, nicht kennen. Versionen des Macro Assemblers, die kleiner als 5.0 sind, unterstützen diese Direktiven jedoch nicht. Mit älteren Assembler-Versionen sollten Sie die Direktiven **SEGMENT, GROUP, ASSUME** und **ENDS** verwenden, die äquivalent zu den vereinfachten Segment-Direktiven sind.

Die folgende Tabelle zeigt die Standard-Segmentnamen, die von jeder Direktive bei dem Medium-Modell erzeugt werden, das einzige für BASIC anwendbare Modell. Die Verwendung dieser Segmente stellt die Kompatibilität mit Microsoft-Sprachen sicher und hilft Ihnen, auf globale Symbole zuzugreifen. Dieser Tabelle folgt eine Liste von drei Schritten, die darstellt, wie die eigentlichen Deklarationen ausgeführt werden müssen.

Direktive	*Name*	*Ausrichtung*	*Kombination*	*Klasse*	*Gruppe*
.CODE	*name*_**TEXT**	**WORD**	**PUBLIC**	**'CODE'**	**---**
.DATA	**_DATA**	**WORD**	**PUBLIC**	**'DATA'**	**DGROUP**
.CONST	**CONST**	**WORD**	**PUBLIC**	**'CONST'**	**DGROUP**
.DATA?	**_BSS**	**WORD**	**PUBLIC**	**'BSS'**	**DGROUP**
.STACK	**STACK**	**PARA**	**STACK**	**'STACK'**	**DGROUP**

Die Direktiven in obiger Tabelle beziehen sich auf die folgenden Segmentarten:

Direktive	*Beschreibung des Segmentes*
.CODE	Das Segment, das den gesamten Code des Moduls enthält.
.DATA	Initialisierte Daten.
.DATA?	Nicht-initialisierte Daten. Microsoft-Compiler speichern nicht-initialisierte Daten separat, weil sie effizienter gespeichert werden können als initialisierte Daten.
.CONST	Konstante Daten. Microsoft-Compiler verwenden dieses Segment für solche Größen wie Zeichenketten- und Gleitkomma-Konstanten.
.STACK	Stapel. Normalerweise ist dieses Segment für Sie im Hauptmodul deklariert und sollte nicht umdeklariert werden.

Die folgenden Schritte beschreiben, wie mit obiger Tabelle Direktiven gebildet werden:

1. Verwenden Sie obige Tabelle, um für Ihre Code- und Daten-Segmente den Segmentnamen, den Ausrichtungstyp, den Kombinationstyp sowie die Klasse festzulegen. Wenn Sie ein Segment definieren, sollten Sie jedes der Attribute verwenden. Das Code-Segment wird zum Beispiel wie folgt deklariert:

   ```
   Name_TEXT        SEGMENT   WORD PUBLIC  'CODE'
   ```

   *Name*_TEXT und alle Attribute werden aus obiger Tabelle entnommen, Sie ersetzen *Name* mit Ihrem Modulnamen. Falls der Kombinationstyp privat ist, verwenden Sie einfach keinen Kombinationstyp.

2. Falls Sie in **DGROUP** Segmente haben, bringen Sie diese mit der Direktive **GROUP** in **DGROUP**, wie in:

   ```
   GROUP     DGROUP     _DATA _BSS
   ```

3. Verwenden Sie **ASSUME** und **ENDS** ganz normal. Beim Eintritt zeigen **DS** und SS beide auf **DGROUP**. Eine Prozedur, die **DGROUP** verwendet, sollte daher die folgende **ASSUME**-Direktive enthalten:

   ```
   ASSUME     CS:TEXT, DS:DGROUP, SS:DGROUP
   ```

Hinweis Wenn Ihre Assembler-Prozeduren reelle Zahlen einsetzen, müssen die Prozeduren Zahlen im IEEE-Format benutzen, um zu QuickBASIC kompatibel zu sein. Dies ist Standard beim Microsoft Macro Assembler, Version 5.0. Mit früheren Versionen müssen Sie die Befehlszeilen-Option **IR** oder die Direktive **8087** angeben.



Anhang D: Zusammenfassung der Befehle

D.1 Aufruf-Optionen

Die folgende Tabelle zeigt und erklärt alle verfügbaren Optionen für den Aufruf von **qb** von der DOS-Befehlszeile aus.

Option	*Beschreibung*
/ah	Ermöglicht es dynamischen Verbunddatenfeldern, Zeichenketten fester Länge sowie numerischen Daten, jeweils größer als 64K zu sein. Falls diese Option nicht angegeben ist, beträgt die maximale Größe pro Datenfeld 64K. Beachten Sie, daß diese Option keine Auswirkung auf die Art hat, mit der Datengrößen an Prozeduren übergeben werden
/b	Startet mit einer CGA in schwarzweiß
/c:*Puffergröße*	Setzt die Größe des Puffers, der Ferndaten empfängt
/cmd *Zeichenkette*	Speichert für die Funktion **COMMAND$** alles ab **/cmd** bis zum Ende der Befehlszeile. Falls vorhanden, muß diese Option die letzte Option auf der DOS-Befehlszeile sein
/g	Veranlaßt QuickBASIC dazu, den Bildwechsel so schnell wie möglich durchzuführen. Wenn Sie diese Option mit einer CGA verwenden, kann "Schnee" auftreten
/h	Zeigt die auf Ihrer Hardware höchstmögliche Auflösung an
/l[*Bibliotheksname*]	Lädt die von *Bibliotheksname* angegebene Quick-Bibliothek
/mbf	Veranlaßt die Konvertierungsfunktionen von QuickBASIC, Zahlen im IEEE-Format als Zahlen im Microsoft-Binär-Format zu behandeln
/run *Quelldatei*	Lädt, kompiliert und startet die angegebene Quelldatei
Quelldatei	Lädt die angegebene Quelldatei

Die folgende Tabelle zeigt und erklärt alle verfügbaren Optionen für den Aufruf von **bc** von der DOS-Befehlszeile aus.

Option	*Beschreibung*
/a	Erzeugt ein Listing, das den Assembler-Code zeigt, den der Compiler für jede Zeile der Quelldatei erzeugt
/ah	Ermöglicht es dynamischen Verbunddatenfeldern, Zeichenketten fester Länge sowie numerischen Daten, jeweils größer als 64K zu sein
/c: *Puffergröße*	Setzt die Größe des Puffers, der Ferndaten empfängt
/d	Erzeugt Debug-Code für Fehlerprüfung während der Laufzeit und schaltet STRG+UNTBR ein
/e	Zeigt an, daß **ON ERROR**-Anweisung zusammen mit einer **RESUME** *Zeilennummer*-Anweisung vorhanden ist
/mbf	Veranlaßt die Konvertierungsfunktionen von QuickBASIC, Zahlen im IEEE-Format als Zahlen im Microsoft-Binär-Format zu behandeln.
/o	Ersetzt die Laufzeit-Bibliothek BRUN40.LIB durch BCOM40.LIB.
/r	Speichert Datenfelder zeilenweise, nicht spaltenweise
/s	Schreibt von Anführungszeichen eingeschlossene Zeichenketten anstelle der symbolischen Tabelle in die Objektdatei. Erfordert somit weniger Speicher.
/v	Schaltet für Datenübertragung (**COM**), Lichtstift (**PEN**), Joystick (**STRIG**), Uhr (**TIMER**), Musik-Puffer (**PLAY**) sowie Funktionstasten (**KEY**) Ereignisverfolgung zwischen BASIC-Anweisungen ein.
/w	Schaltet Ereignisverfolgung für dieselben Anweisungen wie /v ein, überprüft aber zwischen Zeilen auf das Auftreten eines Ereignisses.
/x	Zeigt an, daß **ON ERROR** mit **RESUME, RESUME NEXT** oder **RESUME 0** vorhanden ist.
/zd	Erzeugt Sätze von Zeilennummern, die den Zeilennummern der Quelldatei entsprechen. Diese Sätze sind für die Verwendung mit dem Microsoft SYMDEB symbolischen Debugger vorgesehen.
/zi	Erzeugt Debug-Informationen für den Microsoft CodeView-Debugger.

D.2 Menübefehle und Tastenkurzkombinationen

Die folgende Tabelle zeigt und erläutert die QuickBASIC-Menübefehle und -Tastenkurzkombinationen.

Menü	*Befehl (Tastenkurzkombination)*	*Funktion*
Datei	**Neues Programm**	Entfernt aktuell geladenes Programm aus dem Speicher
	Programm laden	Lädt neues Programm in den Speicher
	Zusammenführen	Fügt Datei in aktuelles Modul ein
	Speichern	Speichert aktuelles Modul als Diskettendatei ab
	Speichern unter	Speichert aktuelles Modul mit dem angegebenen Namen und Format
	Alles Speichern	Schreibt alle geladenen Module in Dateien auf Diskette/Festplatte
	Datei anlegen	Legt neues Modul, neue Include-Datei oder neues Dokument an
	Datei laden	Lädt existierendes Modul, existierende Include-Datei oder existierendes Dokument
	Datei entfernen	Entfernt geladenes Modul, geladene Include-Datei oder geladenes Dokument aus dem Speicher
	Drucken	Druckt markierten Text oder markierte Module
	Betriebssystem	Ruft neue DOS-Oberfläche auf
	Ende	Beendet QuickBASIC
Bearbeiten	**Rückgängig** (ALT+RÜCKTASTE)	Macht letzte Editierung rückgängig
	Ausschneiden (UMSCHALTTASTE+ENTF)	Entfernt Text und kopiert diesen in die Zwischenablage
	Kopieren (STRG+EINFG)	Kopiert Text in die Zwischenablage
	Einfügen (UMSCHALTTASTE+EINFG)	Fügt Inhalt der Zwischenablage an aktueller Position ein

Menü	*Befehl (Tastenkurzkombination)*	*Funktion*
Bearbeiten	**Löschen** (ENTF)	Löscht Text, ohne diesen in die Zwischenablage zu kopieren
	Neue SUB	Öffnet ein Fenster für eine neue **SUB**-Prozedur
	Neue FUNCTION	Öffnet ein Fenster für eine neue **FUNCTION**-Prozedur
	Syntax überprüfen	Schaltet die automatische Syntaxüberprüfung des Editors ein oder aus
Ansicht	**SUBs** (F2)	Gibt eine Liste der geladenen **SUB**-, **FUNCTION**-, Modul-, Include- oder Dokument-Dateien aus
	Nächste SUB (UMSCHALTTASTE+F2)	Zeigt nächstes Unterprogramm im aktiven Fenster an
	(STRG+F2)	Zeigt vorhergehendes Unterprogramm im aktiven Fenster an
	Teilen	Schaltet zwischen geteiltem Bildschirm und einem Arbeitsbereich hin und her
	Nächste Anweisung	Zeigt die nächste auszuführende Anweisung an
	Ausgabebildschirm (F4)	Schaltet auf den Ausgabebildschirm um
	Bearbeiten Include-Datei	Zeigt Zeilen einer Include-Datei zur Editierung an
	Anzeigen Include-Datei	Zeigt Zeilen einer Include-Datei an
	Optionen	Verändert Attribute des Bildschirms
Suchen	**Suchen**	Sucht angegebenen Text
	Markierter Text (STRG+\)	Sucht markierten Text
	Weitersuchen (F3)	Sucht das nächste Auftreten eines angegebenen Textes
	Ändern (STRG+Q A)	Sucht und ersetzt angegebenen Text
	Marke	Sucht Zeilenmarke

Menü	*Befehl (Tastenkurzkombination)*	*Funktion*
Ausführen	**Start** (UMSCHALTTASTE+F5)	Startet aktuelles Programm
	Neustart	Startet aktuelles Programm neu
	Weiter (F5)	Setzt Ausführung fort
	Ändere COMMAND$	Setzt die von der Funktion **COMMAND$** zurückgegebene Zeichenkette
	EXE-Datei erstellen	Erstellt ausführbare Datei auf Diskette
	Bibliothek erstellen	Erstellt Quick-Bibliothek auf Diskette
	Hauptmodul bestimmen	Wechselt Hauptmodul
Debug	**Variable anzeigen**	Setzt Anzeigeausdruck
	Stoppbedingung	Setzt Stoppbedingung
	Anzeigevariable löschen	Löscht den angegebenen Eintrag im Debug-Fenster
	Alle Anzeigevariablen löschen	Löscht alle Einträge des Debug-Fensters
	Verfolgen ein	Hebt die aktuell ausgeführte Anweisung hervor
	Rückverfolgen ein	Zeichnet Ausführungsreihenfolge der Anweisungen auf
	Haltepunkt ein/aus (F9)	Setzt/löscht Haltepunkt an der Cursorposition
	Alle Haltepunkte löschen	Löscht alle Haltepunkte
	Nächste Anweisung festlegen	Kennzeichnet nächste auszuführende Anweisung
Aufrufe	*Prozedur*	Zeigt aktive Prozeduraufrufe an, bewegt den Cursor auf die Aufrufzeile von *Prozedur*
Hilfe	**Allgemein** (F1)	Bietet Informationen zu allgemeinen Themen
	Begriff (UMSCHALTTASTE+F1)	Bietet kontext-sensitive Informationen
	Hilfe beenden (ESC)	Schließt das Hilfefenster

Die folgende Tabelle zeigt und erläutert alle QuickBASIC-Tastenkurzkombinationen sowie die entsprechenden Menübefehle.

Tastenkurzkombination	*Funktion*	*Menübefehl*
F1	Zeigt Informationen zu allgemeinen Themen an	Befehl **Allgemein** aus dem Menü **Hilfe**
F2	Gibt eine Liste der geladenen **SUB-**, **FUNCTION-**, Modul-, Include- oder Dokumentdateien aus	Befehl **SUBs** aus dem Menü **Ansicht**
F3	Sucht nach dem nächsten Auftreten eines zuvor angegebenen Textes	Befehl **Weitersuchen** aus dem Menü **Suchen**
F4	Zeigt Ausgabebildschirm an	Befehl **Ausgabebildschirm** aus dem Menü **Ansicht**
F5	Setzt Programmausführung ab der aktuellen Anweisung fort	Befehl **Weiter** aus dem Menü **Ausführen**
F6	Läßt das nächste Fenster aktives Fenster werden	---
F7	Führt ein Programm bis zur aktuellen Cursorposition aus	---
F8	Führt die nächste Programmanweisung aus; verfolgt durch Prozedur hindurch	---
F9	Schaltet Haltepunkt um	Befehl **Haltepunkt ein/aus** aus dem Menü **Debug**
F10	Führt die nächste Programmanweisung aus; verfolgt um eine Prozedur herum	---
UMSCHALTTASTE+F1	Zeigt kontext-sensitive Informationen zu QuickBASIC-Schlüsselwörtern an	Befehl **Begriff** aus dem Menü **Hilfe**
UMSCHALTTASTE+F2	Zeigt nächstes Unterprogramm im aktiven Fenster an	Befehl **Nächste SUB** aus dem Menü **Ansicht**

Tastenkurzkombination	***Funktion***	***Menübefehl***
UMSCHALTTASTE+F5	Startet Programmausführung vom Beginn an	Befehl **Start** aus dem Menü **Ausführen**
UMSCHALTTASTE+F6	Läßt vorhergehendes Fenster aktives Fenster werden	---
UMSCHALTTASTE+F8	Rückverfolgen nach oben	---
UMSCHALTTASTE+F10	Rückverfolgen nach unten	---
UMSCHALTTASTE+ENTF	Löscht markierten Text, speichert diesen in der Zwischenablage	Befehl **Ausschneiden** aus dem Menü **Bearbeiten**
UMSCHALTTASTE+EINFG	Fügt Inhalt der Zwischenablage an aktueller Cursorposition ein	Befehl **Einfügen** aus dem Menü **Bearbeiten**
STRG+F2	Zeigt vorhergehendes Unterprogramm im aktiven Fenster an	---
STRG+F5	Bringt Fenster, das gesamten Bildschirm einnimmt, auf vorherige Größe zurück	---
STRG+F10	Läßt aktives Fenster gesamten Bildschirm einnehmen/stellt geteilten Bildschirm wieder her	---
STRG+\	Sucht markierten Text	Befehl **Markierter Text** aus dem Menü **Suchen**
STRG+EINFG	Kopiert markierten Text in die Zwischenablage	Befehl **Kopieren** aus dem Menü **Bearbeiten**
STRG+Q A	Sucht und ersetzt markierten Text	Befehl **Ändern** aus dem Menü **Suchen**
ALT+RÜCKTASTE	Macht die letzte Bearbeitung rückgängig	Befehl **Rückgängig** aus dem Menü **Bearbeiten**
ENTF	Löscht markierten Text, speichert diesen nicht in der Zwischenablage	Befehl **Löschen** aus dem Menü **Bearbeiten**
ESC	Schließt Hilfefenster, Dialogfelder, Fenster für Fehlermeldungen oder Menüs	Befehl **Hilfe beenden** aus dem Menü **Hilfe**

D.3 Debug-Befehle

Die zum Debuggen verwendeten Tastaturbefehle werden in der folgenden Tabelle zusammengefaßt.

Funktion	*Taste*
Ausgabebildschirm betrachten	F4
Ausführung ab der aktuellen Anweisung fortsetzen	F5
Startet die Ausführung vom Beginn an	UMSCHALTTASTE+F5
Ausführen bis zur aktuellen Cursorposition	F7
Ausführen der nächsten Programmanweisung; verfolgen durch Prozedur hindurch	F8
Rückverfolgen nach oben	UMSCHALTTASTE+F8
Haltepunkt ein/aus	F9
Ausführen der nächsten Programmanweisung; verfolgen um Prozedur herum	F10
Rückverfolgen nach unten	UMSCHALTTASTE+F10

D.4 Editierbefehle

Die folgende Tabelle faßt die Editierbefehle zusammen.

Cursor bewegen	*Tasten der Tastatur*	*WordStar-Stil*
Zeichen links	NACH LINKS (←)	STRG+S
Zeichen rechts	NACH RECHTS (→)	STRG+D
Wort links	STRG+NACH LINKS (←)	STRG+A
Wort rechts	STRG+NACH RECHTS (→)	STRG+F
Zeile nach oben	NACH OBEN (↑)	STRG+E
Zeile nach unten	NACH UNTEN (↓)	STRG+X
Zeilenanfang	POS1	STRG+Q S
Anfang der nächsten Zeile	STRG+EINGABETASTE	STRG+J
Zeilenende	ENDE	STRG+Q D

Cursor bewegen	*Tasten der Tastatur*	*WordStar-Stil*
Fenster ganz oben	---	STRG+Q E
Fenster ganz unten	---	STRG+Q X
Anfang eines Moduls/ einer Prozedur	STRG+POS1	STRG+Q R
Ende eines Moduls/einer Prozedur	STRG+ENDE	STRG+Q C
Markierungspunkte setzen	---	STRG+K *n**
Zum Markierungspunkt bewegen	---	STRG+Q *n**

Einfügen	*Tasten der Tastatur*	*WordStar-Stil*
Einfügemodus ein- oder ausschalten	EINFG	STRG+V
Zeile darunter	ENDE EINGABETASTE	---
Zeile darüber	---	POS1 STRG+N
Inhalt der Zwischenablage	UMSCHALTTASTE+EINFG	---
Tabulator an der Cursorposition oder Anfang jeder markierten Zeile	TAB	STRG+I
Steuerzeichen an der Cursorposition	---	STRG+P STRG+*Taste***

Löschen	*Tasten der Tastatur*	*WordStar-Stil*
Aktuelle Zeile, in der Zwischenablage speichern	---	STRG+Y
Bis zum Ende der Zeile, in der Zwischenablage speichern	---	STRG+Q Y
Zeichen links	RÜCKTASTE	STRG+H
Zeichen über dem Cursor	ENTF	STRG+G
Wort	---	STRG+T
Markierten Text, nicht in der Zwischenablage speichern	ENTF	STRG+G
Markierten Text, in der Zwischenablage speichern	UMSCHALTTASTE+ENTF	---
Führende Leerzeichen einer Einrückungsebene markierter Zeilen	UMSCHALTTASTE+TAB	---

Kopieren	***Tasten der Tastatur***	***WordStar-Stil***
Markierten Text, in der Zwischenablage speichern	STRG+EINFG	---

Rollen	***Tasten der Tastatur***	***WordStar-Stil***
Eine Zeile nach oben	NACH OBEN (↑)	STRG+W
Eine Zeile nach unten	NACH UNTEN (↓)	STRG+Z
Seite nach oben	BILD ↑	STRG+R
Seite nach unten	BILD ↓	STRG+C
Ein Fenster nach links	STRG+NACH LINKS (←)	---
Ein Fenster nach rechts	STRG+NACH RECHTS (→)	---

Markieren	***Tasten der Tastatur***	***WordStar-Stil***
Zeichen links	UMSCHALTTASTE+ NACH LINKS (←)	---
Zeichen rechts	UMSCHALTTASTE+ NACH RECHTS (→)	---
Aktuelle Zeile	UMSCHALTTASTE+ NACH UNTEN (↓)	---
Zeile darüber	UMSCHALTTASTE+ NACH OBEN (↑)	---
Wort links	UMSCHALTTASTE+ STRG+NACH LINKS (←)	---
Wort rechts	UMSCHALTTASTE+ STRG+NACH RECHTS (→)	---
Bildschirm nach oben	UMSCHALTTASTE+BILD ↑	---
Bildschirm nach unten	UMSCHALTTASTE+BILD ↓	---
Bis zum Anfang eines Moduls/ einer Prozedur	UMSCHALTTASTE+ STRG+POS1	---
Bis zum Ende eines Moduls/ einer Prozedur	UMSCHALTTASTE+ STRG+ENDE	---

* Ersetzen Sie *n* durch eine Zahl von 0 bis 3, um den gewünschten Markierungspunkt zu kennzeichnen.

** Der Kombination STRG+P folgen STRG sowie die gewünschte Taste.

D.5 Fenster-Befehle

Die Tastatur- und Mausversion für jeden Fenster-Befehl werden in der folgenden Tabelle gezeigt. Solange nicht anders angegeben, klicken Sie den linken Knopf, wenn Sie die Maus verwenden.

Funktion	***Tastatur***	***Maus***
Hilfebildschirm aufrufen	F1	Klicken Sie auf den Befehl **Allgemein** aus dem Menü **Hilfe**
Kontext-sensitive Hilfe	UMSCHALTTASTE+F1	Klicken Sie auf den Befehl **Begriff** aus dem Menü **Hilfe**
Vorheriges Suchen wiederholen	F3	Klicken Sie auf den Befehl **Weitersuchen** aus dem Menü **Suchen**
Auf Ausgabebildschirm schalten	F4	Klicken Sie auf den Befehl **Ausgabebildschirm** aus dem Menü **Ansicht**
Arbeitsbereich teilen	---	Klicken Sie auf den Befehl **Teilen** aus dem Menü **Ansicht**
Ausführung fortsetzen	F5	Klicken Sie auf den Befehl **Weiter** aus dem Menü **Ausführen**
Zum nächsten Fenster gehen	F6	Klicken Sie auf das gewünschte Fenster
Zum vorhergehenden Fenster gehen	UMSCHALTTASTE+F6	Klicken Sie auf das gewünschte Fenster
Nächste Prozedur zeigen	UMSCHALTTASTE+F2	Klicken Sie auf den Befehl **Nächste SUB** aus dem Menü **Ansicht**
Vorhergehende Prozedur zeigen	STRG+F2	---
Bis hierhin ausführen	F7 auf Position	Klicken Sie den rechten Knopf an Position
Nächste Anweisung ausführen	F8	---
Haltepunkt hier setzen/löschen	F9 auf Position	Klicken Sie auf den Befehl **Haltepunkt ein/aus** aus dem Menü **Debug**

Funktion	*Tastatur*	*Maus*
Über Prozedur verfolgen	F10	---
Vorhergehende Rückverfolgung anzeigen	UMSCHALTTASTE+F8	---
Im Rückverfolgungsfenster nach unten gehen	UMSCHALTTASTE+F10	---
Läßt aktives Fenster gesamten Bildschirm einnehmen/stellt geteilten Bildschirm wieder her	STRG+F10	Doppelklicken Sie auf der Titelleiste oder klicken Sie auf der Vergrößerungs-Ikone mit dem linken Knopf
In den Menümodus gehen	ALT	---
Aktives Fenster vergrößern	ALT+PLUS	Titelleiste nach oben ziehen
Aktives Fenster verkleinern	ALT+MINUS	Titelleiste nach unten ziehen
Zeile im Fenster nach oben rollen	NACH OBEN (↑)	Klicken Sie auf den Pfeil nach oben
Seite im Fenster nach oben rollen	BILD ↑	---
Rollen zum obersten Fensterbereich	STRG+POS1	---
Zeile im Fenster nach unten rollen	---	Klicken Sie auf den Pfeil nach unten
Seite im Fenster nach unten rollen	BILD ↓	---
Rollen zum untersten Fensterbereich	STRG+ENDE	---
Cursor nach oben bewegen	NACH OBEN (↑)	---
Cursor nach unten bewegen	NACH UNTEN (↓)	---



Glossar

Ziel dieses Glossars ist es, ungewöhnliche und schwierige Begriffe, die in den Handbüchern zu QuickBASIC, Version 4.0, verwendet werden, zu definieren. Weder mit den einzelnen Definitionen noch mit der Liste der Begriffe ist beabsichtigt, ein umfassendes Wörterbuch für die Terminologie der Computer-Wissenschaft bereitzustellen.

Animieren

Eine QuickBASIC-Debug-Möglichkeit, bei der jede Zeile eines laufenden Programmes während derr Ausführung hervorgehoben ist. Der Befehl **Verfolgen ein** aus dem Menü **Debug** schaltet Animation ein.

Anweisung

Eine Kombination von einer oder mehreren Zeilenmarken und BASIC-Schlüsselwörtern, Variablen und Operatoren. Eine BASIC-Anweisung kann allein auf einer einzelnen Programmzeile stehen. Der Ausdruck Anweisung bezieht sich sowohl auf einzeilige Anweisungen, wie zum Beispiel **OPEN** oder **INPUT**, als auch auf eine gesamte aus mehreren Zeilen aufgebaute Anweisung, wie zum Beispiel **IF...END IF**, **SELECT CASE...END SELECT**.

Teile zusammengesetzter Anweisungen, wie zum Beispiel **ELSE** oder **CASE**, werden als "Klauseln" bezeichnet.

Anzeigbar

Ein Debug-Ausdruck, der beschreibt, ob eine Variable oder ein Ausdruck in dem aktuellen Kontext beobachtet werden kann. Eine Größe ist nur anzeigbar, während sich die Ausführung in dem Teil des Programmes befindet, von dem aus Sie die Größe in das Debug-Fenster eingegeben haben.

Siehe "Kontext".

Arbeitsbereich

Der Bereich des Bildschirms, der dazu verwendet wird, Teile einer Quelldatei anzuzeigen oder zu editieren.

Argument

Ein Datum, das einer BASIC-Anweisung, -Funktion oder -Prozedur zur Verfügung gestellt wird. In dem folgenden Beispiel ist die Zeichenkette "Hallo" ein Argument der BASIC-Anweisung **PRINT**:

```
PRINT "Hallo"
```

Ein Argument kann eine Konstante oder Variable sein. Argumente werden häufig dazu verwendet, Informationen aus einem Teil eines Programmes an einen anderen Teil zu übergeben.

Aufruf als Referenz

Siehe "Übergabe als Referenz"

Aufruf als Wert

Siehe "Übergabe als Wert"

Ausführbare Datei

Eine Datei mit der Erweiterung .EXE, .COM oder .BAT. Ausführbare Dateien können gestartet werden, indem Sie den Dateinamen hinter der DOS-Anfrage eingeben.

Ausführen

Ein Programm starten oder eine bestimmte Anweisung im Direkt-Fenster ausführen.

Automatische Variable

Eine Variable in einer **SUB**- oder **FUNCTION**-Prozedur, deren Wert zwischen Aufrufen der Prozedur nicht gesichert wird. Automatische Variablen werden häufig in rekursiven Prozeduren verwendet, wenn Sie die Werte zwischen den Aufrufen nicht sichern lassen möchten. Automatische Variable ist die Standardeinstellung, wenn Sie in der **SUB**- oder **FUNCTION**-Anweisung das Schlüsselwort **STATIC** auslassen.

Basic Input/Output System (BIOS)

In einem Computer eingebaute Programme in Maschinensprache. Diese Programme führen in einem Computer-System die unterste Ebene von Operationen aus. Sie können mit den Anweisungen **CALL** INTERRUPT sowie **CALL** INTERRUPTX direkt auf die BIOS-Programme zugreifen.

Basisname

Der Teil eines Dateinamens vor der Erweiterung. Zum Beispiel ist BEISP der Basisname der Datei BEISP.BAS.

Befehl

Eine Position aus einem QuickBASIC-Menü oder ein DOS-Befehl.

Benutzerbibliothek

Form einer Bibliothek, die ähnlich zu Quick-Bibliotheken ist, aber in früheren QuickBASIC-Versionen verwendet wurde. Benutzerbibliotheken sind nicht kompatibel zu QuickBASIC, Version 4.0.

Siehe "Quick-Bibliothek".

Benutzerdefinierter Typ

Eine zusammengesetzte Datenstruktur, die sowohl Zeichenketten als auch numerische Variablen enthalten kann, ähnlich zu den "Strukturen" von C oder den Records von Pascal. Benutzerdefinierte Typen werden mit **TYPE**-Anweisungen definiert.

Die Datenstruktur wird mit einer **TYPE...END TYPE**-Anweisung definiert. In dem folgenden Code-Ausschnitt ist `TextTyp` ein benutzerdefinierter Typ, `Text` ist ein Datenfeld aus Verbunden, und ein Element des Datenfeldes `Text`, zum Beispiel `Text(50)`, ist ein Verbund.

```
TYPE TextTyp
   ZeilenPos     AS INTEGER
   SpaltenPos    AS INTEGER
   TextFeld      AS STRING * 80
END TYPE
DIM Text(1000) AS TextTyp
```

Siehe Kapitel 2, "Datentypen", im *BASIC-Befehlsverzeichnis*, sowie Abschnitt 3.4.6.2 in *Programmieren in BASIC: Ausgewählte Themen.*

Bibliothek

Eine Datei, die die Erweiterung .LIB (Library) hat und eine oder mehrere kompilierte Objektdateien enthält. Das Binden eines kompilierten Programms mit einer Bibliothek ermöglicht diesem Programm Zugriff auf alle Prozeduren der Bibliothek.

Siehe "Quick-Bibliothek".

Binden (Link)

Der Schritt, den der Linker ausführt, um eine ausführbare Datei zu erzeugen. Binden löst Bezüge zu Prozeduren oder Variablen in anderen Modulen auf und erzeugt ein vollständiges Programm, das ausgeführt werden kann.

Bindezeit

Die Ausführungszeit des Linkers. Während dieser Zeit faßt und bindet er Objektdateien und Bibliotheksdateien zusammen.

Siehe "Kompilierzeit" sowie "Laufzeit".

Binär

Ein mathematischer Begriff, der sich auf ein Zahlensystem zur Basis zwei bezieht, in dem nur zwei Ziffern verwendet werden: 0 und 1. In Eingabe/Ausgabe-Operationen sind binäre Dateien oder Daten diejenigen, die sowohl nicht-druckbare Zeichen als auch normale ASCII-Zeichen enthalten. Binärer Dateizugriff ist die Fähigkeit, innerhalb einer Datei auf ein beliebiges Byte zuzugreifen.

Blinkendes Unterstreichungszeichen

Siehe "Cursor".

Compiler

Ein Programm, das BASIC-Programme in eine für den Computer verständliche Sprache übersetzt.

Cursor

Die blinkende Linie oder das blinkende Rechteck auf dem Bildschirm. Der Cursor kennzeichnet die aktuelle Position, an der QuickBASIC bzw. Ihr BASIC-Programm Eingaben akzeptiert.

Debug-Fenster

Fenster, das Informationen über Variablen oder Ausdrücke ausgibt, die Sie während des Debuggens verfolgen. Für jede angezeigte Komponente zeigt das Debug-Fenster den Kontext der Komponente, deren Namen sowie ihren aktuellen Wert an, wenn diese Komponente anzeigbar ist.

DEF FN-Funktion

Das Objekt, das mit einer einzeiligen **DEF FN**-Anweisung oder mit einem **DEF FN...END DEF**-Block definiert ist. Frühere BASIC-Versionen verwendeten **DEF FN**-Funktionen, um es Benutzern zu ermöglichen, eigene Funktionen zu definieren, die genauso wie BASIC-Funktionen (zum Beispiel **ABS** und **SQR**) arbeiten.

DEF*Typ*

Eine Bezeichnung, die in diesen Handbüchern für die Gruppe von BASIC-Anweisungen verwendet wird, die den Standard-Variablentyp eines Programmes umdefinieren (**DEFDBL, DEFINT, DEFLNG, DEFSNG, DEFSTR**).

Dialogfeld

Ein Feld, das erscheint, wenn Sie einen Menübefehl wählen, der zusätzliche Informationen erfordert.

Dimension

Die Anzahl an Indizes, die erforderlich sind, um ein einzelnes Datenfeldelement anzugeben. Zum Beispiel deklariert die Anweisung

```
DIM A$ (4,12)
```

ein zweidimensionales Datenfeld, während

```
DIM B$(16,4,4)
```

ein dreidimensionales Datenfeld deklariert.

Direkt-Fenster

Der Bereich im unteren Teil des QuickBASIC-Bildschirms, der dazu verwendet wird, Anweisungen einzugeben und direkt auszuführen.

Dokument

Eine Möglichkeit zur Klassifizierung einer Datei, wenn Sie diese in den QuickBASIC-Editor laden. Das Laden einer Datei als Dokument schaltet die Syntax-Überprüfung, die Großschreibung von Schlüsselwörtern sowie andere Eigenschaften des intelligenten Editors aus. Sie können ein BASIC-Programm als Dokument laden, wenn Sie es so wie mit einem Textsystem editieren möchten.

Doppelte Genauigkeit

Ein reeller (Gleitkomma-) Wert, der acht Bytes im Speicher belegt. Werte mit doppelter Genauigkeit sind auf 15 oder 16 Ziffern genau.

$DYNAMIC

Kennzeichnet ein Datenfeld, das während der Laufzeit zugewiesen wird.

Siehe "**$STATIC**".

Einfache Genauigkeit

Ein reeller (Gleitkomma-) Wert, der vier Bytes im Speicher belegt. Werte einfacher Genauigkeit sind auf sieben Dezimalstellen genau.

Einfache Variable

Eine BASIC-Variable, die kein Datenfeld ist. Einfache Variablen können vom Typ numerisch, Zeichenkette oder benutzerdefiniert sein.

Eingabefixierung

Kennzeichnet eine Position in einem Dialogfeld, die Objekt der nächsten Aktion ist. Die Position, die die Eingabefixierung besitzt, ist hervorgehoben, entweder mit hoher Intensität oder invertierter Darstellung.

Erweiterung

Der Teil eines Dateinamens, der dem Punkt folgt. Zum Beispiel ist .DAT die Erweiterung des Dateinamens TEST.DAT.

Fenster

Ein Ausdruck, der sich auf einen der folgenden Punkte bezieht:

- Ein Bildschirmbereich, der dazu verwendet wird, einen Ausschnitt einer Datei anzuzeigen oder Anweisungen einzugeben. "Fenster" bezieht sich nur auf den Bereich des Bildschirms und nicht auf das, was in diesem Bereich angezeigt wird.

  Siehe "Direkt-Fenster", "Debug-Fenster" und "Arbeitsbereich".
- Der Ausschnitt eines logischen Koordinatensystems, das mit der Grafikanweisung **WINDOW** auf die physikalischen Koordinaten abgebildet wird.

FLAGS

Ein Register, das Informationen über den Status der Zentraleinheit und die letzte ausgeführte Operation enthält.

FUNCTION

Ein Block von Anweisungen, der mit **FUNCTION** beginnt und mit **END FUNCTION** endet und eine Prozedur definiert, die wie die BASIC-Funktionen SQR oder **HEX$** aufgerufen werden kann. **FUNCTION**-Prozeduren können anstelle der in älteren BASIC-Versionen verwendeten **DEF FN**-Funktionen benutzt werden.

Ganzzahl (Integer)

Eine ganze Zahl, die innerhalb der Maschine als eine 16-Bit-Binärzahl im Zweierkomplement dargestellt wird. Eine Ganzzahl liegt im Bereich von -32.768 bis +32.767.

Siehe "Lange Ganzzahl".

Geltungsbereich

Der Bereich von Anweisungen, über den eine Variable oder Konstante definiert ist.

Siehe "Globale Konstante", "Globale Variable", "Lokale Konstante" und "Lokale Variable".

Geradengestaltung

Eine Methode zum Zeichnen unterbrochener oder gepunkteter Geraden mit der Grafikanweisung **LINE**. Dies wird ausgeführt, indem in der **LINE**-Anweisung eine 16-Bit-Gestaltungsmaske angegeben wird.

Gleitkomma

Ähnlich der wissenschaftlichen Schreibweise; stellt eine geeignete Methode dar, sehr große oder sehr kleine Zahlen oder Zahlen mit Brüchen darzustellen. Eine Gleitkommazahl besteht aus zwei Teilen: Einem Exponenten und einem als Mantisse bezeichneten Bruchteil.

Siehe "IEEE-Format" sowie Kapitel 2, "Datentypen", im *BASIC-Befehlsverzeichnis*.

Globale Konstante

Eine Konstante, die überall in einem Modul verfügbar ist. Im Modul-Ebenen-Code definierte symbolische Konstanten sind globale Konstanten.

Globale Variable

Eine Variable, die überall in einem Modul verfügbar ist. Globale Variablen können in **COMMON**-, **DIM**- oder **REDIM**-Anweisungen deklariert werden, indem das Attribut **SHARED** verwendet wird.

Hauptmodul

Das Modul, das den Punkt enthält, an dem die Programmausführung beginnt (der Eingangspunkt des Programmes).

Siehe "Modul".

Heap

Ein Bereich des Direktzugriffspeichers, der von BASIC verwendet wird, um unter anderen Variablen und Datenfelder zu speichern.

Hervorhebung

Ein Bereich in invertierter Darstellung in einem Textfeld, Fenster oder Menü, der den aktuell gewählten Befehl oder Text, der zum Kopieren oder Löschen ausgewählt wurde, markiert.

Siehe "Eingabefixierung".

IEEE-Format

Eine Methode zur internen Darstellung von Gleitkommazahlen. Das IEEE-Format erzeugt genauere Ergebnisse als das in früheren QuickBASIC-Versionen verwendete Microsoft-Binär-Format und vereinfacht den Einsatz eines mathematischen Koprozessors. Die Abkürzung IEEE steht für Institute of Electrical and Electronics Engineers, die Organisation, die diesen Standard entwickelt hat.

Siehe "Gleitkomma".

Include-Datei

Eine Quelldatei, die mit dem Metabefehl **$INCLUDE** in ein Programm kompiliert wird.

Interpreter

Ein Programm, das Anweisung für Anweisung übersetzt und ausführt. Ein Interpreter übersetzt eine Anweisung jedesmal, wenn er die Anweisung ausführt. BASICA ist ein Sprachen-Interpreter.

Vergleiche mit "Compiler".

Kacheln

Das Ausfüllen einer Figur mit einem Muster anstelle einer durchgängigen Farbe, indem die Grafikanweisung **PAINT** verwendet wird.

Klausel

Teil einer BASIC-Anweisung. In

```
IF NOT Gefunden(SchlWort$) THEN
   FehlerKennz = WAHR
   CALL FehlerProz("Kein Schlüsselwort")
ELSE
   CALL InTabelle(SchlWort$)
END IF
```

ist der folgende Ausschnitt eine Klausel:

```
ELSE
   CALL InTabelle(SchlWort$)
```

Klicken

Betätigen und loslassen eines der Mausknöpfe, während mit der Maus auf ein Objekt auf dem Bildschirm gezeigt wird.

Kompilieren

Der Vorgang, den QuickBASIC oder BC durchführen, um BASIC-Anweisungen in eine Form zu übersetzen, die vom Computer ausgeführt werden kann.

Kompilierzeit

Die Zeit, während der BASIC-Anweisungen übersetzt werden. Solche Fehler, wie falsche Syntax, werden während der Kompilierung entdeckt und als Kompilierzeitfehler bezeichnet.

Konstante

Ein Wert, der sich während der Programmausführung nicht ändert. Im Gegensatz dazu ist eine Variable ein Wert, der sich während der Programmausführung ändern kann – und sich normalerweise auch ändert.

Siehe "Symbolische Konstante"

Kontext

Befindet sich auf der QuickBASIC-Statusleiste, um den ausführenden Teil eines laufenden Programmes zu kennzeichnen. Dieser Kontext besteht aus dem Namen eines Moduls, oder dem Namen eines Moduls, gefolgt von dem Namen einer Prozedur.

Kontrollfeld

Ein Feld, das es Ihnen ermöglicht, Optionen ein- oder auszuschalten. Wenn die Option eingeschaltet ist, erscheint ein X in dem Feld.

Kurze Adresse (Near Address)

Ein Speicherbereich, der nur mit dem Offset vom Beginn des Segmentes an angegeben wird. Eine kurze Adresse erfordert nur zwei Bytes.

Siehe "Lange Adresse".

Laden

Eine Datei in den Speicher kopieren.

Lange Adresse (Far Address)

Ein Speicherbereich, der angegeben wird, indem ein Segment (Bereich eines Blockes mit 64K) und ein Offset vom Beginn des Segmentes an verwendet werden. Lange Adressen erfordern vier Bytes - zwei für das Segment und zwei für den Offset. Eine lange Adresse wird auch als Intersegmentadresse bezeichnet.

Lange Ganzzahl

Eine ganze Zahl, die innerhalb der Maschine als eine 32-Bit-Binärzahl im Zweierkomplement dargestellt wird. Eine lange Ganzzahl liegt im Bereich von -2.147.483.648 bis +2.147.483.647.

Siehe "Ganzzahl".

Laufzeit

Die Zeit, während der ein Programm ausgeführt wird. Laufzeit bezieht sich auf die Ausführungszeit eines Programmes und nicht auf die Ausführungszeit des Compilers oder des Linkers. Einige Fehler - als Laufzeitfehler bezeichnet - können nur entdeckt werden, während das Programm läuft.

Laufzeit-Modul

Ein Modul, das die meisten zur Implementierung der BASIC-Sprache notwendigen Routinen enthält. Eine Besonderheit des Laufzeit-Modules besteht darin, daß es eine ausführbare (.EXE) Datei ist. Das Laufzeit-Modul heißt BRUN40.EXE und ist größtenteils eine Bibliothek von Routinen.

Listenfeld

Ein Feld, das eine Serie von Positionen auflistet. Zum Beispiel ist das Feld, das Dateien im Dialogfeld **Programm laden** auflistet, ein Listenfeld.

Siehe "Dialogfeld".

Logische Koordinaten

Zahlenpaare (zum Beispiel *x,y*), die einen beliebigen Punkt eines zweidimensionalen Raumes eindeutig definieren. Die Zahlen *x* und *y* können jede reelle Zahl sein. Ein Satz logischer Koordinaten kann mit der Grafikanweisung **WINDOW** auf dem Bildschirm abgebildet werden. Logische Koordinaten ermöglichen es Ihnen, Daten mit Koordinaten zu zeichnen, die der Kurve oder dem Bild angepaßt sind, ohne sich Gedanken über die aktuellen Bildschirmkoordinaten machen zu müssen.

Siehe "Physikalische Koordinaten".

Lokale Konstante

Eine Konstante, deren Geltungsbereich auf eine Prozedur (**FUNCTION** oder **SUB**) eines Moduls begrenzt ist. Alle innerhalb einer Prozedur in **CONST**-Anweisungen deklarierten Konstanten sind lokale Konstanten.

Lokale Variable

Eine Variable, deren Geltungsbereich auf eine bestimmte Code-Einheit begrenzt ist, wie zum Beispiel Modul-Ebenen-Code oder eine Prozedur (**FUNCTION** oder **SUB**) eines Moduls. In einer Prozedur sind alle Variablen lokal, es sei denn, sie erscheinen in einer **SHARED**-Anweisung innerhalb der Prozedur oder in einer **COMMON SHARED**-, **DIM SHARED**- oder **REDIM SHARED**-Anweisung auf der Ebene des Moduls, das die Prozedurdefinition enthält.

Siehe "Modul-Ebenen-Code" sowie "Prozedur".

Markieren

Positionen in einem Dialogfeld, zum Beispiel Optionen, Listenfelder oder Textfelder, auswählen. "Markieren" bezeichnet ebenso den Vorgang, mit dem Text- oder Grafikbereiche zur Bearbeitung ausgewählt werden.

Maschinencode

Eine Abfolge binärer Zahlen, die der Mikroprozessor als Programmanweisungen ausführt.

Mathematischer Koprozessor

Eine optionale Hardware-Komponente, wie zum Beispiel ein 8087- oder 80287-Chip, die die Geschwindigkeit und Genauigkeit von Berechnungen mit Gleitkommazahlen erhöht. QuickBASIC greift automatisch auf einen mathematischen Koprozessor zurück, falls dieser vorhanden ist, oder emuliert ihn per Software, falls er nicht vorhanden ist.

Siehe "Gleitkomma".

Maus

Ein Zeigegerät, das in Ihre Hand paßt und auf einer glatten Unterlage in jede Richtung bewegt werden kann. Mit der Bewegung der Maus können Sie den Mauszeiger auf dem Bildschirm in eine entsprechende Richtung bewegen.

Siehe "Zeiger".

Menüleiste

Die Leiste im oberen Teil des QuickBASIC-Bildschirms, die die Menübezeichnungen enthält.

Metabefehle

Spezielle Befehle einer Quelldatei, die in Kommentarzeichen eingeschlossen sind und dem Compiler mitteilen, bestimmte Aktionen durchzuführen, während er das Programm kompiliert. Zum Beispiel weist der Metabefehl **$INCLUDE** den Compiler an, Anweisungen aus einer anderen Datei einzufügen.

Microsoft-Binär-Format

Eine Methode zur internen Darstellung von Gleitkommazahlen. Das Microsoft-Binär-Format wird von früheren QuickBASIC-Versionen verwendet.

Siehe "Gleitkomma" sowie "IEEE-Format".

Modul

Eine separierte Gruppe von Anweisungen. Jedes Programm hat mindestens ein Modul (das Hauptmodul). In den meisten Fällen ist ein Modul dasselbe wie eine Quelldatei. Wenn Sie ein Programm speichern, das mehrere Module enthält, wird jedes Modul in einer separaten Diskettendatei gespeichert.

Modul-Ebenen-Code

Programmanweisungen innerhalb eines Moduls, die außerhalb einer **SUB**- oder **FUNCTION**-Definition stehen. Fehler- oder Ereignisbehandlungscode sowie deklarierende Anweisungen, wie zum Beispiel **COMMON**, **DECLARE** und **TYPE**, dürfen nur auf der Modul-Ebene erscheinen.

Objektdatei

Eine Datei (mit der Erweiterung .OBJ), die verschiebbaren Maschinencode enthält, der beim Kompilieren eines Programmes erzeugt wird, und den der Linker dazu verwendet, eine ausführbare Datei zu erstellen.

Objektmodul

Der Inhalt einer Objektdatei, nachdem die Datei Teil einer selbständigen (.LIB) Bibliothek geworden ist.

Siehe "Bibliothek".

Offset

Die Anzahl an Bytes vom Beginn eines Segmentes im Speicher bis zu einem bestimmten Byte in diesem Segment.

Öffnen

Die Operation, die von der BASIC-Anweisung **OPEN** mit einer Datei durchgeführt wird. Die Anweisung **OPEN** bereitet eine Datei für Lesen oder Schreiben vor.

Parameter

Eine Variable in einer **DEF FN-**, **FUNCTION-** oder **SUB**-Anweisung. Parameter werden durch tatsächliche Variablen und Werte ersetzt, wenn Sie die Funktion oder das Unterprogramm aufrufen.

Physikalische Koordinaten

Die Bildschirmkoordinaten. Die physikalischen Koordinaten (0,0) beziehen sich auf die obere linke Ecke des Bildschirms, es sei denn, Sie haben mit einer **VIEW**-Anweisung ein grafisches Darstellungsfeld definiert, so daß in diesem Falle (0,0) die Koordinaten der linken oberen Ecke des Darstellungsfeldes sind.

Der Bereich der physikalischen Koordinaten hängt von dem Bildschirmadapter, dem Monitor sowie den von der Anweisung **SCREEN** gesetzten Spezifizierungen ab.

Siehe "Logische Koordinaten".

Programm

Ein oder mehrere Module, die eine Abfolge von Anweisungen bilden, die eine bestimmte Aufgabe lösen.

Siehe "Modul".

Prozedur

Ein allgemeiner Begriff für eine **SUB** oder **FUNCTION**.

PUBLIC

Eine Direktive des Macro Assemblers, die dazu verwendet wird, Symbole zu deklarieren, die in mehr als einem Modul verwendet werden. PUBLIC-Symbole sind wichtig in mehrsprachigen Programmen, die Assembler-Prozeduren enthalten. Sie müssen die Namen aller Prozeduren, die Sie aus BASIC heraus aufrufen möchten, sowie die Namen von Daten, die Sie zwischen BASIC- und Assembler-Abschnitten gemeinsam benutzen möchten, als PUBLIC deklarieren. BASIC-**SUB**- und **FUNCTION**-Prozeduren sind implizit global (public).

Quelldatei

Eine Textdatei, die BASIC-Anweisungen enthält. Obwohl alle Module Quelldateien sind, sind nicht alle Quelldateien (Z.B. Include-Dateien) Module.

QuickBASIC verwendet Quelldateien mit zwei unterschiedlichen Formaten. Quelldateien, die im QuickBASIC-Format gespeichert sind, werden schnell geladen, können aber nicht direkt mit einem normalen Texteditor gelesen oder verändert werden. Textdateien (oder ASCII-Dateien) können mit jedem Texteditor gelesen oder verändert werden.

Quick-Bibliothek

Eine Datei, die auf eine der folgenden Arten erzeugt wurde:

- Mit dem Befehl **Bibliothek erstellen** aus dem QuickBASIC-Menü **Ausführen**
- Mit der Option **/Q** des Befehls **link**

Eine Quick-Bibliothek enthält Prozeduren, die von mehr als einem Programm verwendet werden können. Quick-Bibliotheken wurden in früheren QuickBASIC-Versionen als "Benutzerbibliotheken" bezeichnet. Benutzerbibliotheken können nicht als Quick-Bibliotheken eingesetzt werden, und umgekehrt.

Register

Bereiche des Prozessors, in denen Daten während der Verarbeitung auf Maschinenebene temporär gespeichert werden. Die in der 8086-Prozessorfamilie verwendeten Register sind: AX, BX, CX, DX, SP, BP, SI, DI, DS, ES, SS, CS, IP und das Register FLAGS.

Siehe "FLAGS".

Rolleisten

Die Leisten, die auf der rechten Seite sowie im unteren Teil des Arbeitsbereiches bzw. einiger Listenfelder erscheinen. Rolleisten erlauben es Ihnen, den Inhalt eines Arbeitsbereiches oder eines Textfeldes mit der Maus zu rollen.

Rollen

Den Text nach oben bzw. nach unten oder nach links bzw. rechts bewegen, um Textteile zu sehen, die nicht mehr auf den Bildschirm passen.

Schlüsselwort

Der Name einer BASIC-Anweisung, -Funktion oder eines BASIC-Operators. Beispiele für Schlüsselworte sind **MOD**, **OPEN**, **OR**, **PRINT** und **SIN**.

Seite

Siehe "Fenster".

Selbständige Datei

Eine ausführbare Datei, die ohne Laufzeitdatei (ein Programm, das nach der Kompilierung mit BCOM40.LIB gebunden wurde) gestartet werden kann.

Speicherabbild (Memory Map)

Eine Darstellung, an welcher Speicherstelle der Computer bestimmte Typen von Informationen erwartet.

Standard-Datentyp

Der Typ, den BASIC für numerische Daten verwendet, es sei denn, Sie geben mit einer **DEF***Typ*-Anweisung einen anderen Typ an. Der Standardtyp für numerische Daten ist "einfache Genauigkeit".

Stapel (Stack)

Ein Bereich des Direktzugriffspeichers, der von BASIC verwendet wird, um Objekte wie Zwischenergebnisse arithmetischer Berechnungen, automatische Variablen in Prozeduren sowie an Prozeduren übergebene Argumente, zu speichern. Der Stapel wird von BASIC außerdem zum Sichern solcher Informationen, wie zum Beispiel, wo eine Prozedur aufgerufen wurde, eingesetzt.

Starten

Ein Programm ausführen.

$STATIC

Kennzeichnet ein Datenfeld, das während der Kompilierzeit zugewiesen wird.

Siehe "Automatische Variable", "**$DYNAMIC**" sowie "**STATIC**"; siehe außerdem Anhang C, "Metabefehle", im *BASIC-Befehlsverzeichnis*.

STATIC

Eine lokale Variable in einer **SUB** oder **FUNCTION**, deren Wert zwischen Aufrufen gesichert wird.

Statusleiste

Die unterste Zeile des QuickBASIC-Bildschirms. Die Statusleiste zeigt Ihre Textposition, den Tastaturstatus, den aktuellen Kontext der Ausführung sowie weitere Programminformationen an.

SUB

Eine Code-Einheit, die von den Anweisungen **SUB** und **END SUB** eingeschlossen ist. Eine **SUB** (manchmal als "Unterprogramm" bezeichnet) bietet eine Alternative zu dem älteren BASIC-Unterroutinentyp **GOSUB**.

Symbolische Konstante

Eine Konstante, die von einem Symbol und nicht der Konstanten selbst dargestellt wird. Symbolische Konstanten werden mit der Anweisung **CONST** definiert und ermöglichen es, ein Programm lesbar bzw. veränderbar zu gestalten.

Textfeld

Ein Feld, in das Sie Informationen eingeben, die für die Ausführung eines Befehls benötigt werden. Ein Textfeld erscheint innerhalb eines Dialogfeldes. Was Sie eingeben, erscheint in dem Feld. Das Textfeld kann leer sein, wenn das Dialogfeld erscheint, oder das Feld kann Text enthalten, wenn es eine Standard-Option gibt, oder Sie etwas für diesen Befehl zutreffendes gewählt haben.

Titelleiste

Eine Leiste im oberen Arbeitsbereich, die den Namen des Moduls (und der Prozedur, falls zutreffend) zeigt, das aktuell in diesem Fenster angezeigt wird.

Übergabe als Referenz (Passing by Reference)

Es wird die Adresse eines Argumentes an eine **SUB** oder **FUNCTION** übergeben. Übergabe als Referenz ermöglicht es einer Prozedur, die ihr übergebenen Argumentenwerte zu verändern.

Übergabe als Wert (Passing by Value)

Übergibt den Wert (und nicht die Adresse) eines Argumentes an eine **DEF FN-**, **FUNCTION-** oder **SUB**-Anweisung.

Überlauf

Tritt auf, wenn der einer numerischen Variablen zugewiesene Wert außerhalb des für diesen Variablentyp zulässigen Bereichs liegt.

(Um)Schalter, (um)schalten

Eine Funktionstaste oder Menüwahl, die eine Eigenschaft ausschaltet, wenn sie eingeschaltet ist, oder eine Eigenschaft einschaltet, wenn sie ausgeschaltet ist. Zum Beispiel ist der Befehl **Verfolgen ein** aus dem Menü **Debug** ein Umschalter, der die animierte Anzeige einer Verfolgung ein- oder ausschaltet.

Ungelöster Verweis

Ein Symbol, auf das in einem Assembler-Modul Bezug genommen wird, welches aber in einem anderen Modul, das mit dem Assembler-Modul gebunden ist, nicht als PUBLIC erklärt ist. Ungelöste Verweise werden normalerweise durch Schreibfehler oder dadurch verursacht, daß der Name des Moduls, das die gewünschten Symbole enthält, auf der link-Befehlszeile ausgelassen ist.

Unterlauf

Eine Fehlerbedingung, die verursacht wird, wenn eine Berechnung ein Ergebnis hervorbringt, das für die interne Behandlung im Computer zu klein ist.

Unterprogramm

Siehe "SUB".

Unterroutine

Eine Einheit mit BASIC-Code, die von der Anweisung **RETURN** beendet wird. Die Programmkontrolle wird mit einer **GOSUB**-Anweisung an eine Unterroutine übergeben.

Verbund (Record)

Ein Begriff, der sich auf einen der folgenden Punkte bezieht:

- Eine Variable, die mit einem benutzerdefinierten Typ deklariert ist.
  Siehe "Benutzerdefinierter Typ".
- Mehrere Felder, behandelt als zusammengehörende Einheit in einer Datendatei.

Wählen

Einen Befehl ausführen, indem zum Beispiel eine Befehlsfläche aus einem Dialogfeld oder eine Position aus einem Menü ausgesucht wird.

Welt-Koordinaten

Siehe "Logische Koordinaten".

Zeichenkette (String)

Eine zusammenhängende Folge von Zeichen, häufig gekennzeichnet mit einem symbolischen Namen. In diesem Beispiel ist "Hallo" eine Zeichenkette:

```
PRINT "Hallo"
```

Eine Zeichenkette kann eine Konstante oder Variable sein. QuickBASIC 4.0 unterstützt Zeichenketten fester und variabler Länge.

Zeiger

Das invertiert dargestellte Rechteck, das sich bewegt, um die aktuelle Position der Maus zu kennzeichnen. Der Mauszeiger erscheint nur, wenn eine Maus installiert ist. Um eine Position mit der Maus zu wählen, bewegen Sie die Maus, bis der Zeiger auf der Position verbleibt, und betätigen anschließend den linken Knopf.

Ziehen

Die Maus auf ein Objekt im Bildschirm zeigen lassen, einen Mausknopf drücken, und die Maus anschließend bewegen, während der Knopf gedrückt bleibt.

Zweierkomplement

Eine Möglichkeit zur Darstellung ganzer Zahlen innerhalb der Maschine. Positive Zahlen werden als direkte Binärzahlen dargestellt. Negative Zahlen werden als Zweierkomplement des positiven Wertes dargestellt.

Siehe Kapitel 2, "Datentypen", im *BASIC-Befehlsverzeichnis*.

Zwischenablage (Clipboard)

Ein Speicherbereich, der Text enthält, den Sie kopieren oder verschieben. In der Zwischenablage wird Text nicht gesammelt: Neuer Text, der der Zwischenablage hinzugefügt wird, löscht jeden Text, der sich dort bereits befindet.

Index

B

R

S

X

Z

Microsoft®

0188 Artikelnr. 01648

Microsoft®

0188 Artikelnr. 01648

Microsoft® QuickBASIC 4

Lernen und Anwenden von Microsoft QuickBASIC

Microsoft® QuickBASIC 4

Lernen und Anwenden von Microsoft QuickBASIC

Microsoft®